VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vysokozátěžové úložiště grafů

Heavy-load graph data repository

2010 Bc. Jiří Hudeček

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. května 2010 .....

Abstrakt

Obsahem této diplomové práce je snaha zvýšit efektivitu ukládání grafových dat získaných aplikací GuardControl firmy ORBIT s.r.o. Zvýšení efektivity bude mít za následek menší paměťové nároky a rychlejší odezvu při zobrazování historie výsledků testů. Cílem bude také zlepšit grafický výstup, což povede k lepší vypovídací schopnosti grafů.

Klíčová slova: graf, uložení grafů v databázi

Abstract

The content of this thesis is to increase the efficiency of storing graph data obtained by the GuardControl application created by the ORBIT Company Ltd. Increased efficiency will result in less memory requirements and improved response time for displaying the history of test results. Second goal is improving graphical output, resulting in better explanatory power of the graphs.

Keywords: graph, chart, storing graph in database

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

GuardControl[1] je monitorovací systém pro ICT infrastrukturu. Jeho úkolem je sledovat IT prostředky, předvídat možné problémy, informovat o nastalých chybách a reagovat na ně. V tuto chvíli se stará přibližně o 800 serverů celkem od 20 klientů. Postupně se zvyšující počet klientů používajících GuardControl znamená stále více testovaných strojů a tedy i více grafových dat. Dnes je prováděno přibližně 10 000 testů, z nichž přes 6 400 ukládá grafová data. Obrovský objem ukládaných grafových dat odhalil nedostatek, který na malém vzorku není patrný - data zaberou hodně místa a režie s nimi spojená je pomalá. V následujících kapitolách rozeberu problém podrobněji, vysvětlím, v čem spočívá moje řešení a zhodnotím dosažené výsledky.

2 Definice problému

Pro lepší pochopení problému nejprve popíšu současný způsob ukládání grafových dat.

- Testovací služba spustí test.
- Po skončení testu se výsledek uloží do tabulky *GraphPoints*.
- V půlhodinových intervalech se spouští uložené procedury, které naměřené hodnoty průměrují a průměry ukládají do tabulek *GraphData_5m*, *GraphData_30m*, *GraphData_2h* a *GraphData_12h*.
- Při každém startu testovací služby a dále pravidelně jednou za 24 hodin se z tabulky *GraphPoints* odstraní původní naměřené hodnoty starší než 24 hodin.
- Zprůměrovaná data se replikují z klientské do centrální databáze.

Struktura databázových tabulek *GraphPoints* a *GraphData_5m* je zobrazena v tabulkách 1 a 2. Tabulky *GraphData_30m*, *GraphData_2h* a *GraphData_12h* mají stejnou strukturu jako tabulka *GraphData_5m*.

2.1 Velikost dat

Zpracování grafových dat se stalo paměťově velmi náročné. Vzorek databáze z období dlouhého 124 dní zabírá 15,8 GB. Z tohoto objemu zabírají grafová data přibližně 83%. Hlavní problémy současného řešení jsou zkreslení dat a velká spotřeba paměti. Vysoké paměťové nároky se projevují také pomalejší odezvou serveru a nutností přenášet velké objemy dat po síti, ať už mezi databázovým serverem a vykreslovací logikou, nebo mezi klientskou a centrální databází v případě replikace dat. Existuje několik faktorů, které se na tomto problému podílejí:

- Databáze obsahuje velké množství řádků. Zmiňovaný vzorek databáze obsahuje přes 91 000 000 řádků v tabulce *GraphData_5m*.
- Jednotlivé řádky v databázi se často liší pouze v několika sloupcích, ostatní zůstávají stejné.
- Tabulky s průměry za delší období sice urychlují přístup k datům, ale zabírají značný prostor.
- Některé typy testů mají jako svůj výsledek hodnotu *true/false*. Pro tyto testy se do grafových dat ukládají prázdné řádky se samými nulami, které jsou zbytečné.

2.2 Vypovídací schopnost grafů

Do databázových tabulek se ukládají čtyři typy průměrovaných hodnot (průměry za 5 minut, 30 minut, 2 hodiny a 12 hodin). Čím více hodnot průměr obsahuje, tím více data zkresluje. Největší problém tedy nastává u vzdálenějších pohledů. Na obrázku 1 je

Název sloupce	Datový typ	Popis
TestID	int	ID testu
TimePoint	datetime	čas výsledku testu
Counted_5m	bit	bity označující, že hodnota již byla započítána do jednotlivých průměrů
Counted_30m	bit	
Counted_2h	bit	
Counted_12h	bit	
Value1	float	hodnoty výsledku testu
Value2	float	
Value3	float	
Value4	float	
Error	bit	bit označující, že nastala chyba
DropOut	bit	bit označující plánovaný výpadek

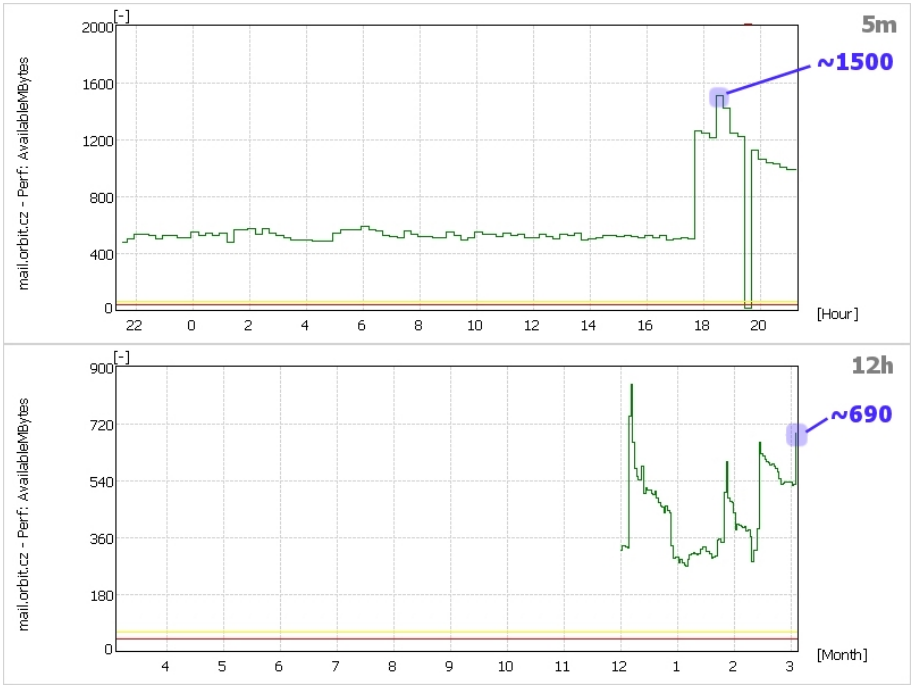
Tabulka 1: Struktura tabulky GraphPoints

srovnání dvou různých pohledů na stejná data. První graf je vykreslen z pětiminutových průměrů a ukazuje interval 24 hodin, druhý z dvanáctihodinových průměrů a ukazuje interval jednoho roku. Konkrétní označená hodnota má díky dvanáctihodinovým průměrům méně než poloviční hodnotu. Pokud by se i vzdálená data četla z průměrů pětiminutových, potřeba zobrazit v grafu velké množství dat by nejspíše vedla opět k průměrování a výsledek by byl velmi podobný. Rozdíl by byl v tom, že by získání a vykreslení dat trvalo déle. Pokud tedy chceme ukládat a vykreslovat jenom průměry, více tabulek se vzdálenějšími průměry nabízí vyšší rychlost zpracování za cenu větších nároků na uložení dat. Pokud chceme celkově zlepšit vypovídací schopnost grafů, samotné průměrování dat stačit nebude. Za špatný vizuální výsledek mohou tyto faktory:

- Ukládání dat je ztrátové - přímé výsledky se ukládají pouze 24 hodin, dále jsou k dispozici už pouze průměrované hodnoty.
- V případě chyby se do průměrů ukládá informace o chybě, okolní data jsou ztracena.
- Webová verze se od desktopové velmi liší - ovládání je jiné, výsledné grafy mají naprosto odlišný vzhled.

Název sloupce	Datový typ	Popis
TestID	int	ID testu
TimePoint	datetime	čas výsledku testu
ReplNeed	bit	bit označující potřebu replikace řádku do centrální databáze
ReplTime	datetime	datum a čas replikace řádku
ValuesCount	bigint	počet zprůměrovaných hodnot
SumValue1	float	součty hodnot z daných intervalů - slouží pro přepočet při přidávání nových hodnot
SumValue2	float	
SumValue3	float	
SumValue4	float	
Value1	float	průměrné hodnoty z daného intervalu
Value2	float	
Value3	float	
Value4	float	
Error	bit	bit označující, že nastala chyba
DropOut	bit	bit označující plánovaný výpadek

Tabulka 2: Struktura tabulky GraphData_5m



Obrázek 1: Srovnání pětiminutových a dvanáctihodinových průměrů

3 Návrh řešení

Cílem návrhu nové reprezentace grafových dat je odstranit oba současné největší problémy - snížit potřebný prostor pro ukládání dat a zároveň zachovat všechna naměřená data bez jejich průměrování. Pro dosažení cíle navrhuji provést následující výčet úprav.

3.1 Ukládání zbytečných dat

Testy, jejichž výsledek nabývá hodnot pouze *true* nebo *false* sice do databáze ukládají grafová data, ale v podobě prázdných řádků se samými nulami. Na testovaném vzorku tato data zabírají přibližně 5% řádků v grafových tabulkách. Nabízí se tedy dvě možnosti - buďto tato data neukládat a ušetřit tak místo, nebo začít ukládat opravdové hodnoty a ty nějakým vhodným způsobem vykreslovat. Po domluvě s vedoucím této práce se již nadále nebudou tyto řádky ukládat.

3.2 Průměrování dat

Při prvotním návrhu vznikla myšlenka průměrovat data z důvodu rychlejšího přístupu ke vzdálenějším pohledům. V takovýchto případech se zpravidla toleruje redundance způsobující větší prostor nutný pro uložení dat.

GaurdControl nabízí celou řadu testů, pro které dlouhodobější průměrování výsledků znamená ztrátu vypovídací schopnosti. Například přes den během pracovní doby bude nejspíše počet přihlášených uživatelů vyšší než v noci. Průměrováním hodnot těchto dvou oblastí dochází ke zkreslování hodnot jak z období pracovní doby, tak i v nočních hodinách. Dlouhodoběji průměrovaná data se tedy stávají zbytečná a není proto důvod je dále ukládat. Jako součást řešení tedy navrhuji místo stávajících čtyř tabulek s průměrovanými daty používat tabulku jedinou, ve které se budou ukládat všechna data. Na testovaném vzorku zabírají „zbytečné“ tabulky přibližně 28% prostoru. Pro vykreslování grafů se pokusím navrhnout takové možnosti zobrazení, které zvýší vypovídací schopnost výsledku.

3.3 Nová reprezentace dat

Reprezentace dat současným způsobem spojená se zachováním všech naměřených hodnot by měla katastrofální následky. Stovky miliónů řádků by zabíraly desítky gigabytů a systém by byl velmi pomalý. Z tohoto důvodu plyne nutnost uložit do jednoho řádku více hodnot. Vytvářet pro tento účel spoustu pevných sloupců není rozumné, takže poslední zbývající možností je uložit více hodnot do jedné buňky jednoho řádku. Nabízí se například možnost uložit data ve formě XML dokumentu [2]. Tato možnost ovšem nemá šanci na úspěch, protože by nejspíše vyžadovala ještě více místa, než kolik vyžaduje aktuální způsob. Proto bude nejvýhodnější data do řádků uložit v binární podobě.

Současná reprezentace dat potřebuje pro uložení jednoho řádku 392 nebo 520 bitů. Rozdíl je v tom, jestli test ukládá jednu nebo dvě hodnoty. Při uložení více dat do jednoho

řádku se sníží celkový počet řádků, čímž se eliminují vlivy sloupců, které nejsou nositeli hodnot. Tím se ušetří místo za opakující se data:

- *TestID* bude stejné pro jakékoli množství souvisejících dat, tedy za každý ušetřený řádek ušetříme 32 bitů.
- Každý řádek obsahuje režii *NULL* hodnot o velikosti 32 bitů[1], které opět ušetříme s menším počtem řádků.
- Hodnoty *ReplNeed* a *ReplTime* nemají vazbu na hodnotu, ale ke konkrétnímu řádku. Za každý ušetřený řádek tedy ušetříme 64 bitů za sloupec *ReplTime*. Hodnota *ReplNeed* se kvůli vnitřní reprezentaci datového typu *bit* v databázi nijak neprojeví na výsledné úspoře[2].
- Pokud nebudeme hodnoty průměrovat, sloupce *SumValue1 - SumValue4* se stanou zbytečné. Tím se ušetří 64 bitů za každý sloupec, který nese hodnotu.
- Úspora dat na ostatních sloupcích bude závislá na konkrétní reprezentaci dat.

Binární reprezentaci více hodnot je možno navrhnout více způsoby. Podstatou každého řádku je říct, co a kdy se stalo. Pokud tedy do jednoho řádku místo jednoho „co“ a jednoho „kdy“ vložíme binárně stokrát „co“ a stokrát „kdy“, ušetříme místo za 99x *TestID*, 99x *ReplTime*, 99x *SumValue1* a 99x řežie *NULL* hodnot. Tento konkrétní příklad by tedy způsobil úsporu 48% prostoru pro uložení dat.

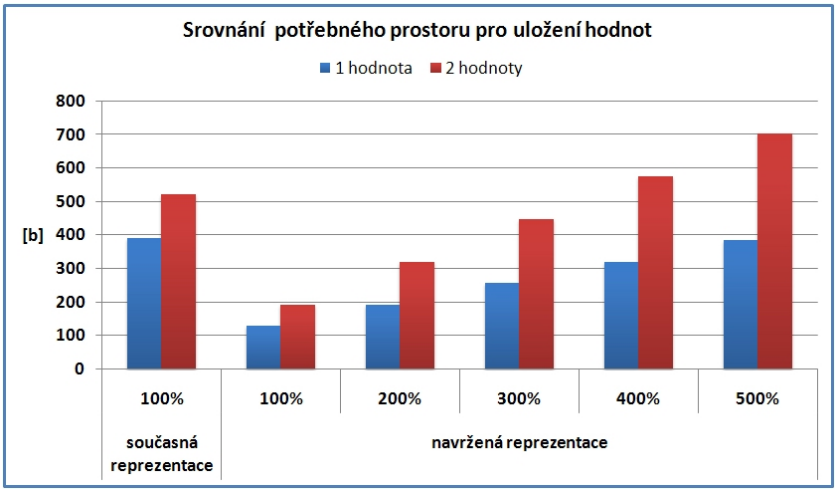
Do jediného řádku by se dalo vložit obrovské množství hodnot (nejen sto) a záleželo by pouze na stanovené hranici počtu hodnot na řádek. Příliš malá hranice by neuspořila moc místa, příliš velká by zbytečně zatěžovala při vykreslování krátkých intervalů.

Řádek v takto navržené tabulce by měl svoji určitou velikost a každá nová hodnota by již zabrala 64 bitů pro datový typ *datetime* a 64 bitů pro každou ukládanou hodnotu *float*. Pro test, který ukládá pouze jednu hodnotu, by nová hodnota zabrala pouze 128 bitů, místo současných 392. Úspora prostoru by tedy činila až 67% při zachování stejného počtu dat. Problém je v tom, že nová verze má za cíl uchovávat větší objem dat, než verze stará a v tu chvíli se úspora místa snižuje. Jak je vidět z grafu na obrázku 2, pokud by nová verze ukládala pětkrát více hodnot než verze současná (tedy průměrný test by byl spouštěn v minutových intervalech), zabrala by tabulka téměř stejný prostor jako nyní, pro dvě ukládané hodnoty by to bylo dokonce více.

Nastíněné řešení stále nabízí prostor pro úsporu místa. Jednotlivé testy jsou spouštěny v pravidelných intervalech, čehož lze využít a místo ukládání všech TimePointů stačí uložit pouze první TimePoint a krok mezi nimi. Pravidelnou inkrementací kroku se dají všechny TimePointy dopočítat. Pokud by se perioda mezi testy z jakéhokoli důvodu změnila, je třeba začít na novém řádku. Zbavením se TimePointů ušetříme na každé nové hodnotě 64 bitů. Počet řádků ovšem již nebude závislý na stanovené hranici počtu hodnot na řádek, ale bude také třeba vytvářet nové řádky v okamžiku, kdy dojde k

[1] Režie v bytech spojená s ukládáním hodnot *NULL* se počítá jako 2 + ((pocet_sloupcu + 7) / 8), viz [3]

[2] SQL server optimalizuje ukládání datového typu *bit* viz [4]



Obrázek 2: Závislost potřebného prostoru na počtu uložených hodnot

přerušení posloupnosti. Servery většinou vykazují stabilní chování, takže takových případů nebude v poměru k celku mnoho.

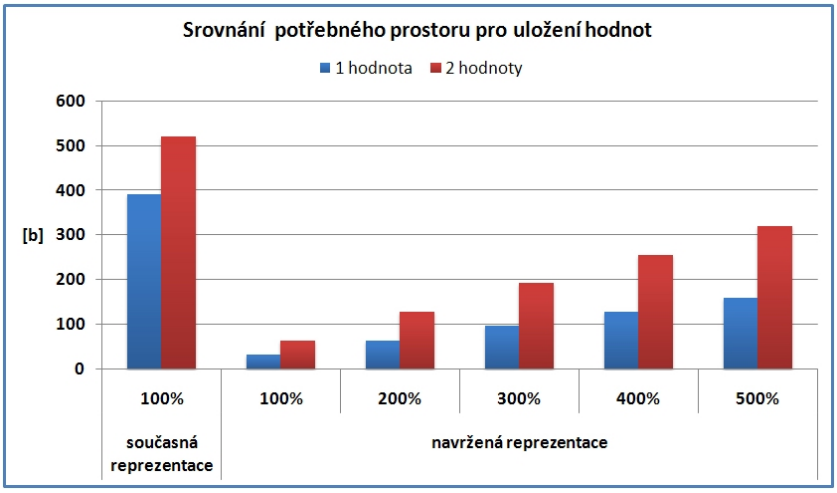
Poslední šance na snížení celkového objemu dat je zmenšení posloupností hodnot. Většina hodnot se nachází v rozumném rozmezí a proto bychom mohli hodnoty do posloupností uložit jako datový typ *int*. Pokud by přece jen naměřená hodota byla vyšší, musela by být uložena samostatně. Obrázek 3 ukazuje předešlý graf po úspoře 64 bitů za typ *datetime* a 32 bitů za každý použitý *int*. Po těchto opatřeních je již výsledek uspokojivý. Stejný počet hodnot zabere přibližně 10% prostoru, pětinásobek hodnot zhruba polovinu.

Hodnoty uložené v posloupnosti čísel mohou být mnohdy relativně malé. Nemůžeme-li však jednoznačně určit maximální hodnotu, není možno použít menší datový typ a ušetřit tak místo. Na posloupnost hodnot by se ale dal použít vhodný kompresní algoritmus, který dosáhne ještě lepšího výsledku.

3.4 Komprese dat

Na zmenšení objemu dat posloupnosti hodnot jsem vybral tzv. Fibonacciho kódování [5]. Tento algoritmus se neřadí přímo mezi kompresní, jedná se pouze o speciální reprezentaci čísel. Pro tuto příležitost se hodí hned z několika důvodů:

- Algoritmus je přímo určený pro kódování posloupnosti přirozených čísel.
- Je vhodný zejména pro kódování malých čísel - tato výhoda zatím není zjevná, bude vysvětlena v následujícím textu.
- I relativně malý počet hodnot dokáže uložit na menší prostor než by vyžadovala sekvence hodnot typu *int*. Oproti tomu například slovníkové kompresní algoritmy mohou mít na malém množství dat za následek větší výstup než vstup.



Obrázek 3: Závislost potřebného prostoru na počtu uložených hodnot

```
0 1 1 2 3 5 8 13 21 34 55 ...
```

Obrázek 4: Fibonacciho posloupnost

V následujících sekcích vysvětlím princip kódování a provedu úpravy potřebné pro použitelnost v tomto projektu.

3.4.1 Princip Fibonacciho kódování

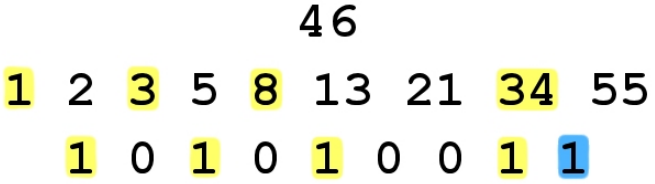
Italský matematik Leonardo Fibonacci je tvůrcem známé, tvz. Fibonacciho posloupnosti [6]. Tato nekonečná posloupnost začíná čísly 0, 1 a každé následující číslo je součtem předchozích dvou. Začátek posloupnosti je zobrazen na obrázku 4.

Pomineme-li první dvě čísla (0, 1), vzniknou váhy použitelné pro kódování jakéhokoli přirozeného čísla. Obrázek 5 ukazuje posloupnost použitelnou pro kódování. Stejně jako u binárního kódování sčítáme váhy o velikosti mocnin dvou (1,2,4,8,16...), u Fibonacciho kódování sčítáme váhy Fibonacciho posloupnosti.

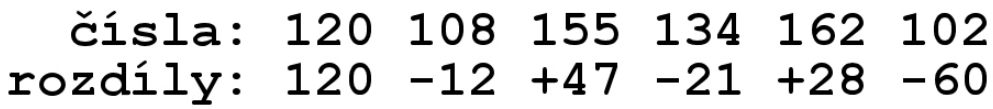
Na první pohled si lze všimnout, že váhy ve srovnání s binárními vahami nerostou příliš rychle. Maximální hodnota, kterou můžeme uložit pomocí klasického binárního kódování do 32 bitů je 4 294 967 295 ($2^{32} - 1$). Oproti tomu Fibonacciho kódování je do tohoto prostoru schopno zapsat největší možné číslo 3 524 577. Fibonacciho kódování má ovšem jednu unikátní vlastnost - v žádném z čísel se v součtu nevyskytují dvě po

```
1 2 3 5 8 13 21 34 55 89 ...
```

Obrázek 5: Fibonacciho posloupnost použitelná pro kódování



Obrázek 6: Příklad kódování čísla 46



Obrázek 7: Přepočet na rozdíly hodnot

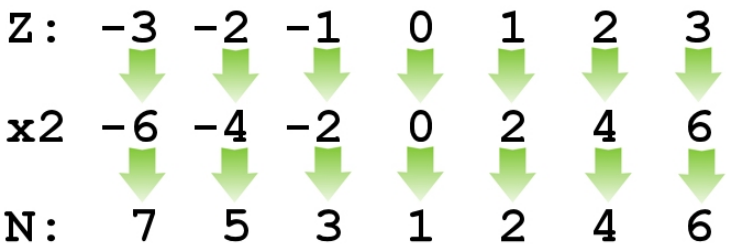
sobě následující váhy. Analogicky pro bity to znamená, že žádné číslo nebude mít ve své bitové reprezentaci dvě jedničky za sebou. Každá váha je tvořena součtem předchozích dvou, proto se místo dvou po sobě jdoucích vah použije jejich součet, který následuje. Připojíme-li za (jedničkou končící) číslo další jedničku, vznikne jednoznačný oddělovač, za nímž může následovat další číslo. Binární kódování oddělovače nepoužívá, a proto musí mít každé číslo pevně stanovenou délku, nehledě na to, jak je ve skutečnosti velké. Například číslo 46 lze zakódovat pomocí 9 bitů. Při použití binárního kódování by stačilo bitů 8, z nichž dva bity by byly zcela nevyužité a byli bychom zároveň omezeni maximální hodnotou 255. Abychom limit zvýšili, nejspíše bychom zvolili bitů více, typicky 32. Na obrázku 6 je příklad zakódování čísla 46. Poslední bit je přidán jako oddělovač.

Nejmenší zakódovatelné číslo 1 zabere pouze dva bity - jeden bit pro hodnotu a druhý jako oddělovač. Pokud je každé číslo uloženo do proměnlivé délky bitů, a tato délka závisí na velikosti čísla, je výhodné, aby ukládaná čísla byla co nejmenší. Rozhodl jsem se proto posloupnost přepočítat na rozdíly mezi jednotlivými hodnotami. Tato metoda je základem tzv. Delta kódování [7]. Testy, měřící například volné místo na disku, velikost souborů nebo počty přihlášených uživatelů, vykazují v po sobě jdoucích výsledcích často velmi malé rozdíly. Ve skutečnosti jen velmi málo testů vykazuje výrazně kolísavé chování. Přepočtem na rozdíly hodnot tedy můžeme dosáhnout výrazného zmenšení velikostí hodnot. Srovnání posloupnosti před a po přepočítání na rozdíly ukazuje obrázek 7.

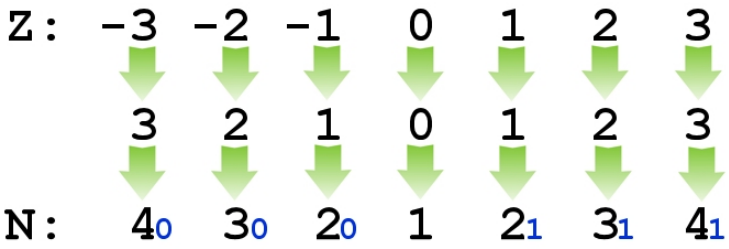
Je zřejmé, že přepočet na rozdíly hodnot má za následek přiblížení se k nule. Ovšem přepočtem na rozdíly nutně musí vzniknout čísla jak kladná, tak i záporná. Jelikož Fibonacciho posloupnost sestává z přirozených čísel, není v základní formě použitelná pro kódování čísel celých.

3.4.2 Potřebné úpravy

Pro zajištění schopnosti ukládat všechna celá čísla existují nejméně dvě možné modifikace základního algoritmu.



Obrázek 8: Mapování čísel střídavým algoritmem



Obrázek 9: Reprezentace čísel algoritmem se znaménkovým bitem

První možností je celá čísla mapovat na čísla přirozená, a to střídavě vždy jedno kladné a jedno záporné. Toho lze dosáhnout pomocí následujících kroků: Vstupní číslo se nejprve vynásobí dvěma. Pokud výsledek není kladný, vynásobí se (-1) a přičte se 1. Ve výsledku jsou záporná čísla reprezentována čísly lichými a kladná sudými. Výsledné mapování ukazuje obrázek 8.

Druhý možný způsob je přidat na konec čísla znaménkový bit. Aby bylo možné vyjádřit i nulu, je nutné nejprve přičíst 1. Pro reprezentaci nuly nebude potřeba ukládat znaménko. Způsob ukazuje obrázek 9, modře jsou zobrazeny znaménkové bity.

Oba algoritmy jsem naimplementoval, a poté otestoval velikost výstupu pro všechna čísla z intervalu $< -1000000, 1000000 >$ Výsledek je vyobrazen v tabulce 3

Celkem čísel	2 000 001
Oba algoritmy stejný počet bitů	1 168 014
Znaménkový algoritmus o bit méně	831 985
Střídavý algoritmus o bit méně	2

Tabulka 3: Srovnání výsledků obou úprav

Z výsledků plyne, že střídavý algoritmus dosahuje na postatné části intervalu horších výsledků. Proto bude pro ukládání dat do databáze použit algoritmus se znaménkovým bitem.

3.5 Kódování posloupností

Data testů mohou mít různý charakter a ne vždy bude možné hodnoty uložit pomocí Fibonacciho kódování. Můžeme například požadovat větší přesnost uložených dat, než jakou poskytují celá čísla. Spolu s posloupností se proto bude ukládat ještě i informace o tom, jak jsou data v posloupnosti reprezentována. To proto, aby bylo jasné, jak data dekódovat. Navhnul jsem tyto typy kódování:

- 0 - None: Není použito pro ukládání hodnot, ale pouze pro lepší orientaci ve zdrojových kódech aplikací.
- 1 - TrueFalse: Toto kódování není ve skutečnosti implementováno, je pouze rezervováno pro případ, že by se v budoucnu tato data ukládala.
- 2 - Integer: Nejefektivnější a zároveň nejvíce používané kódování, kódují se celá čísla výše popsaným upraveným Fibinacciho kódováním.
- 3 - OneDecimal: Kódování čísel s přesností na jedno desetinné místo. Hodnoty se vynásobí desíti a uloží se jako Integer. Po rozkódování se desíti vydělí.
- 4 - Double: Kódování čísel typu double (64-bitové číslo s pohyblivou desetinnou čárkou). V tomto případě dochází pouze k ukládání jednotlivých hodnot do posloupnosti za sebou.

V tuto chvíli se všechny existující testy dělí pouze do skupin s kódováním *None*, *TrueFalse* a *Integer*. Kódování *OneDecimal* je připraveno pro použití v testech, kde je potřeba zachovat přesnější výsledek. Kódování *Double* se již využívá ve chvíli, kdy je naměřená hodnota příliš vysoká. Hranici jsem stanovil na 10^9 pro kódování Integer a 10^8 pro OneDecimal. Při překročení této hranice se hodnota uloží samostatně jako Double.

3.6 Vykreslování dat

Se změnou reprezentace dat je potřeba změnit i vykreslování těchto dat. Ovládání současných verzí si není moc podobné a desktopová verze dosahuje graficky úplně jiného výsledku než webová. Desktopová verze navíc v určitých situacích nefunguje tak, jak by měla, případně jak by uživatel očekával. Mým cílem po stránce vykreslování dat jsou tedy tyto úkoly:

- Zlepšení grafického vzhledu grafu.
- Uživatelská přívětivost ovládacího rozhraní.
- Přidání nových možností zobrazení pro zlepšení vypovídací schopnosti.
- Maximální sjednocení vzhledu a ovládání desktopové a webové verze.

4 Implementace

Pro implementaci budou použity technologie a prostředky, pomocí kterých se GuardControl vyvíjel dosud:

- C# 2.0 [9]
- ASP.NET 2.0 [8]
- MS SQL server 2005 [10]
- Visual Studio 2005 [11]

Novinkou bude rozšíření webové verze o:

- ASP.NET AJAX 1.0 [12]
- AJAX Control Toolkit v. 1.0.20229 [13]

Implmentace je rozdělena do několika částí, které jsou uvedeny v následujících sekcích.

4.1 Změny v databázi

Prvním krokem k dosažení výsledku je provedení změn v databázi - přidání nových tabulek a funkčnosti. Toto vše je obsaženo ve skriptu *DatabaseChanges.sql*. V tabulce 4 je zobrazena nová reprezentace naměřené hodnoty.

Je vidět, že tabulka se od stávajícího řešení se příliš neliší. Ze čtyř bitů označujících započítání do průměrů se stal bit jeden (*Counted*) a přibyl sloupec *LaunchTime*, který určuje, na kdy byl test naplánován. Tato hodnota je důležitá, protože se inkrementuje o stále stejnou hodnotu. *TimePoint* se oproti tomu liší v závislosti na rychlosti provedení testu. Pro možnost zaokrouhlení času výsledku na plánovaný čas byl stanoven limit 30 sekund. Pokud bude mít test větší zpoždění, musí být uložen na samostatný řádek. Zpoždění se děje většinou ve dvou situacích - při extrémní zátěži měřeného serveru, nebo při jeho vypnutí (test je naplánován, ale provede se až po zapnutí serveru). Servery se často nevypínají a extrémně zatížené nebývají, proto nových řádků, vzniklých kvůli zpoždění testů, nebude mnoho.

Tabulka 5 ukazuje, jakým způsobem budou uloženy přímo samotné posloupnosti hodnot. Opět je vidět podobnost s tabulkou *GraphData_5m*, pouze přibyly sloupce popisující posloupnost - kdy posloupnost začíná a končí, o kolik sekund se hodnoty inkrementují, jak jsou zakódovány a zda potřebují spojit s předchozí posloupností. Z tabulky zmizely bity *Error* a *Dropout*, které jsem zařadil do posloupností jako speciální hodnoty:

- 1000 - Error - chyba
- 1001 - Dropout - plánovaný výpadek
- 1002 - NoValue - prázdná hodnota

Název sloupce	Datový typ	Popis
TestID	int	ID testu
TimePoint	datetime	čas výsledku testu
LaunchTime	datetime	čas, na který byl naplánován start testování
Value1	float	hodnoty výsledku testu
Value2	float	
Value3	float	
Value4	float	
Error	bit	bit označující, že nastala chyba
DropOut	bit	bit označující plánovaný výpadek
Counted	bit	bit označující, že hodnota již byla zpracována

Tabulka 4: Struktura tabulky GraphPointsNew

Speciální hodnoty mají vazbu na výsledek testu, takže nemohly zůstat v řádku, ale musely být zařazeny do posloupností. Jelikož neexistují univerzální hodnoty, kterých výsledky testů, přepočítané na rozdíly mezi hodnotami, nemohou nabývat, přiřadil jsem jim hodnoty 1000-1002. Pokud je naměřená hodnota větší nebo rovna 1000, musí se k ní přičíst 3, aby vznikl prostor pro speciální hodnoty. Analogicky, pokud je hodnota v posloupnosti větší než 1002, musí se 3 odečíst, abychom dostali původní hodnotu.

Kromě nových tabulek skript přidá do databáze tyto funkce:

- **sp_Graphs_SetCounted(@TestID int, @TimepointFrom datetime, @TimepointTo datetime)** - uložená procedura pro označení hodnot ze zadané oblasti, které již byly zařazeny do posloupnosti.
- **fn_GetValuesRate(@TestID int,@Border int)** - funkce vracející poměr hodnot překračujících stanovenou hranici ku celkovému počtu.
- **fn_IsOrgUnitLinked(@TestID int)** - funkce zjišťující, zda orgranizační jednotka, pod kterou test spadá, je svázaná s centrální databází (jestli se její data replikují).
- **fn_GetNewGraphPoints(@TestID int)** - funkce vracející tabulku nových, dosud nezpracovaných hodnot.
- **fn_GetGraphTests()** - funkce vracející seznam testů, které ukládají grafová data.
- **GraphDataConvert** - assembly obsahující uložené procedury pro konverzi dat.
- **sp_GraphDataConvert** - procedura projádějící konverzi nových dat z tabulky *GraphPointsNew* do tabulky *GraphData*.

Název sloupce	Datový typ	Popis
TestID	int	ID testu
From	datetime	datum a čas počátku posloupnosti
To	datetime	datum a čas konce posloupnosti
ReplNeed	bit	bit označující potřebu replikace řádku do centrální databáze
ReplTime	datetime	datum a čas replikace řádku
Value1	varbinary(MAX)	posloupnosti jednotlivých hodnot
Value2	varbinary(MAX)	
Value3	varbinary(MAX)	
Value4	varbinary(MAX)	
Step	int	krok - počet sekund mezi jednotlivými hodnotami v posloupnosti
ValueType	tinyint	typ použitého kódování
ValuesCount	int	počet hodnot v posloupnosti
MergeNeed	bit	bit označující, že posloupnost se musí sloučit s jinou (pokud se sloučit dá)

Tabulka 5: Struktura tabulky GraphData

- **sp_GraphDataConvertOld** - procedura konvertující staré pětiminutové průměry z tabulky *GraphData_5m* do tabulky *GraphData*.
- **sp_GraphDataConvertOldTest @TestID int, @TestType int** - procedura konvertující stará data z tabulky *GraphData_5m* pouze pro jeden zvolený test.
- **sp_GraphDataMerge** - procedura provádějící spojování posloupností.
- **sp_SetDropouts @TestID int** - procedura ukládající do tabulky *GraphPointsNew* informace o probíhajícím plánovaném výpadku.
- **sp_TestGraphData @TestID int** - procedura srovnávající stará data z tabulky *GraphData_5m* s novými daty z tabulky *GraphData*.

Použité SQL příkazy, které nejsou přidány do databáze, se nacházejí přímo ve zdrojových kódech aplikací, zpravidla z důvodu dynamického vytváření dotazu.

4.2 Změny v ukládání dat

Změny v přímém ukládání dat plynou ze struktury tabulky *GraphPointsNew*. Výsledná data budou uložená v posloupnostech v tabulce *GraphData*. Ke konverzi naměřených dat do posloupností dochází hned třemi různými způsoby:

1. Jednorázová konverze starých dat při přechodu na novou verzi
2. Pravidelné konverze starých dat replikovaných od klientů, kteří ještě nemají nasazenou novou verzi
3. Pravidelné konverze nových naměřených dat

Každý ze způsobů se liší v konkrétní implemetaci, algoritmus ukládání dat do posloupností však zůstává ve své podstatě stejný. Obrázek 10 ukazuje převádění dat do posloupností, zobrazeny jsou pouze nejdůležitější bloky. V diagramu není uvedeno ověřování maximální velikosti posloupností a ukládání prázných hodnot, tzv. *NoValues*. Ve chvíli, kdy se změní perioda mezi naměřenými hodnotami, se nejprve ověří, jestli nová perioda není dělitelná starou periodou a pokud ano, vyplní se nová perioda práznými hodnotami. Díky tomu není potřeba v situacích, kdy chybí data, vytvářet nové řádky, ale stačí doplnit několik prázných hodnot. Využití bude častější u konverze pětiminutových průměrů, u neprůměrovaných dat budou mít chybějící hodnoty spíše za následek nepravidelnou změnu periody.

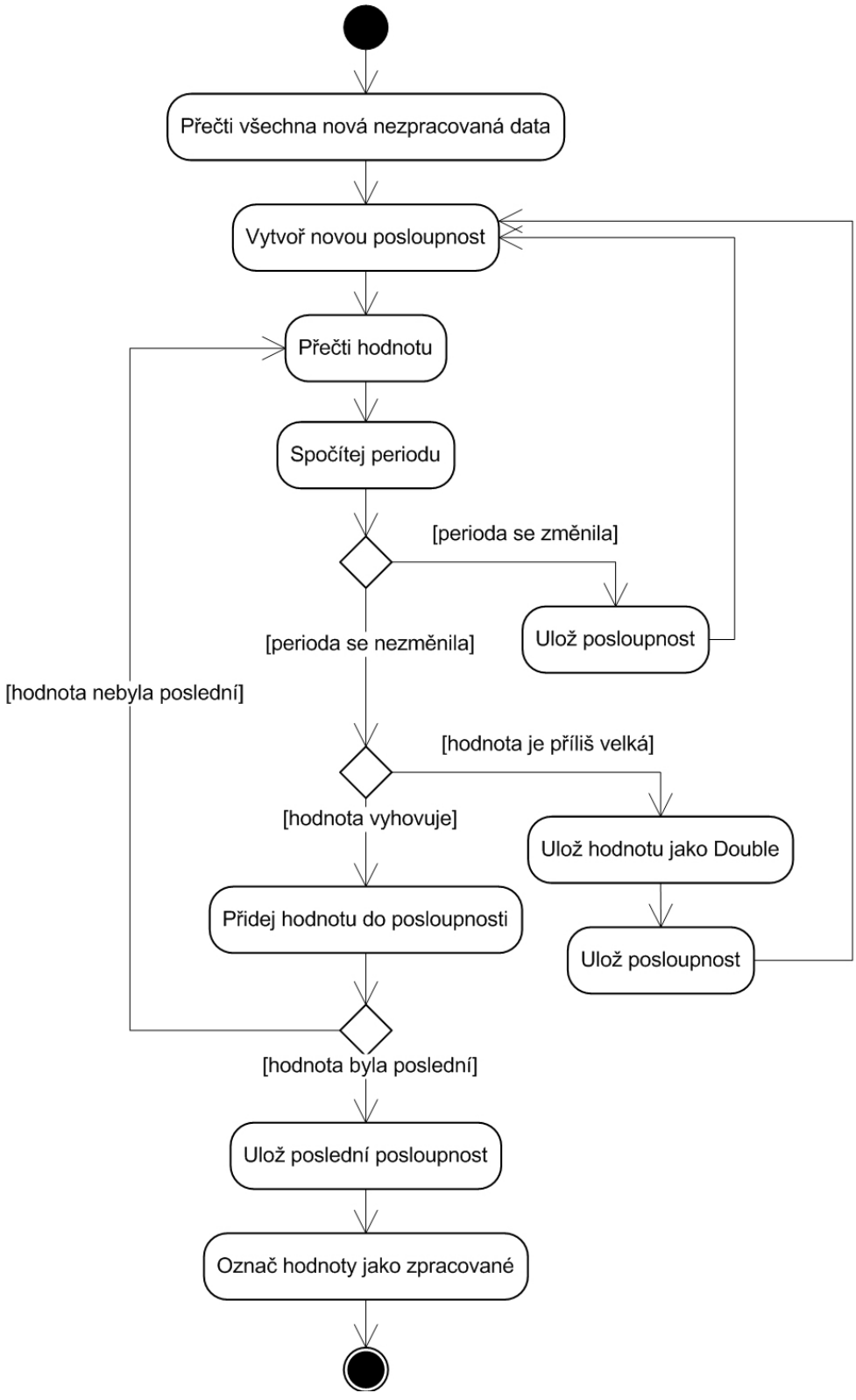
Vytvářené posloupnosti se nesnaží připojit k již existujícím posloupnostem, ale vytváří posloupnosti nové. Tepve až jsou nové posloupnosti replikovány do centrální databáze, dojde k jejich spojení. Díky tomu se po síti přenáší jenom přírůstky dat. Diagram na obrázku 11 ukazuje postup spojování posloupností dohromady.

4.3 Omezení velikosti posloupností

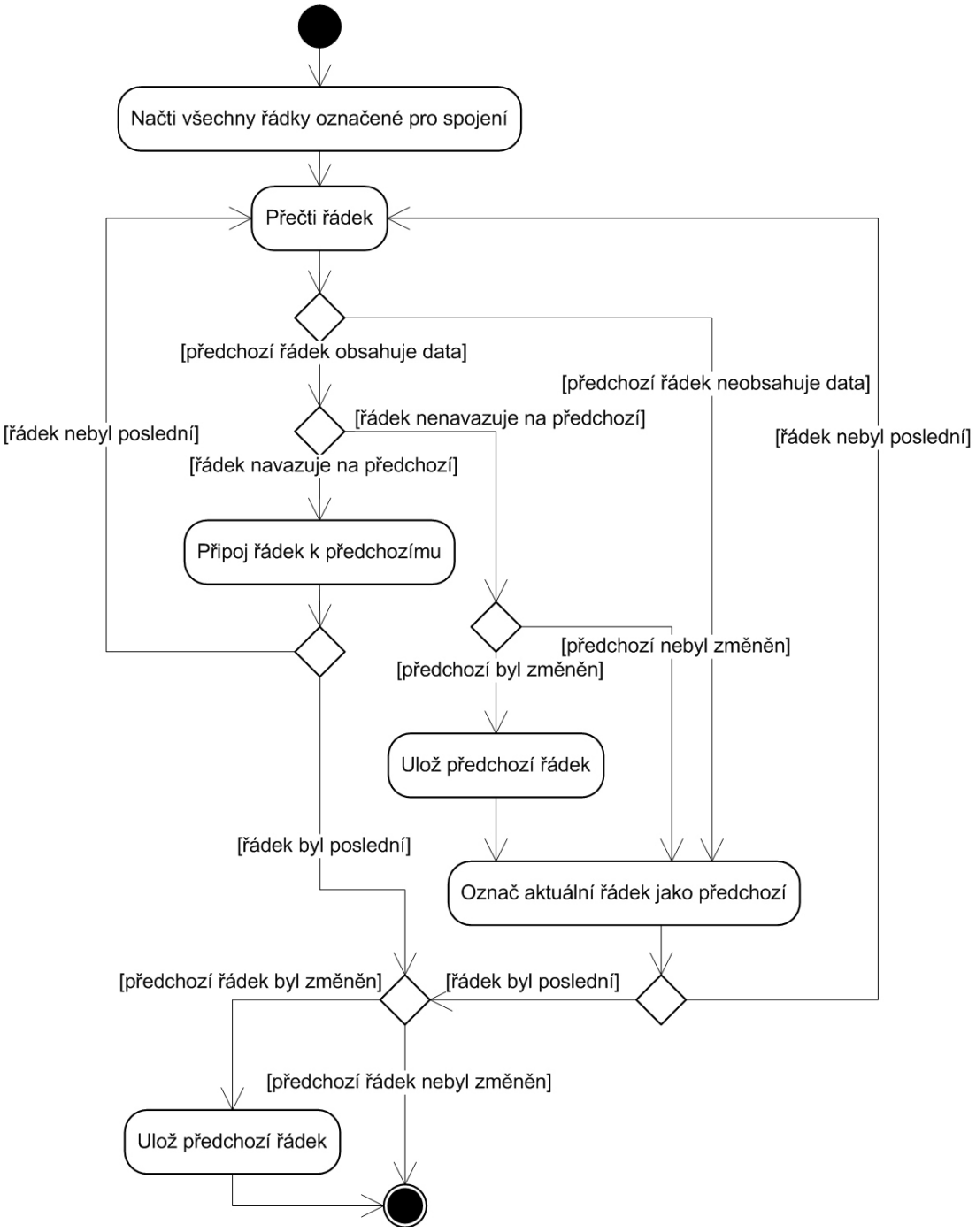
Posloupnost čísel sice může být libovolně dlouhá, ale nevhodnou volbou maximální délky můžou data zabírat zbytečně více prostoru. Pro řádky v databázi platí omezení maximálně 8060 bytů na řádek [14]. Omezení se nevztahuje mimo jiné na datový typ `VARBINARY(MAX)` [15], který je použit pro ukládání posloupností. Dokud celková velikost řádku nepřesáhne limit, bude posloupnost uložena přímo v řádku. Větší posloupnosti jsou uloženy samostatně mimo řádek a na jejich místech v řádku jsou informace o jejch umístění. Vzniklá režie zabírá v databázi místo navíc, což jsem si ověřil jednoduchým pokusem.

Do databáze jsem různými způsoby ukládal 400 000 000 32-bitových čísel seřazených do binární podoby. Po každém měření jsem data smazal a příkazem SHRINK DATABASE [16] jsem zmenšil datový soubor na minimální velikost, což by mělo být dostatečné pro dosažení objektivních výsledků. Výsledek pokusu je v tabulce 6. Z tabulky je patrné, že pokud se data vejdou do řádku, mají očekávanou velikost. Při překročení limitu velikost dat naroste v závislosti na vnitřním způsobu uložení dat databázovým serverem.

Abych zamezil zbytečnému ukládání dat, pokusím se limit stanovit tak, aby se všechna data vešla do řádku. Tohoto stavu není těžké dosáhnout při použitém kódování *Double*. Jedna hodnota zabírá 8 bytů, 1000 hodnot pak 8000 bytů. Do zbylých 60 bytů se vejdou ostatní data v řádku. Pro kódování *Integer* a *OneDecimal* to předem stanovit nelze. Velikost čísla je závislá na jeho hodnotě a není proto jednoznačné, kolik hodnot se do 8000 bytů vejde. Na testovaném vzorku je průměrná velikost jednoho čísla pomocí nové reprezentace 10 bitů. Pokud bych připočítal 2 bity jako rezervu, mohl bych limit



Obrázek 10: Diagram konverze dat do posloupností



Obrázek 11: Diagram spojování posloupností

Počet řádků	Velikost dat	Celková velikost tabulky
800 000	2 kB	1 600 000 kB
400 000	4 kB	1 600 000 kB
200 000	8 kB	1 600 000 kB
160 000	10 kB	1 716 208 kB
100 000	16 kB	1 605 936 kB
80 000	20 kB	1 605 728 kB
50 000	32 kB	1 604 224 kB
40 000	40 kB	1 603 864 kB
25 000	64 kB	1 801 184 kB
20 000	80 kB	1 640 466 kB
12 500	128 kB	1 700 600 kB
10 000	160 kB	1 681 408 kB

Tabulka 6: Prostor pro uložení dat podle jejich rozdělení

stanovit na 5300 hodnot na řádek. Limit jsem tedy stanovil na 5000 hodnot na řádek pro kódování *Integer* a 3000 pro *OneDecimal*.

4.4 Související úpravy a nástroje

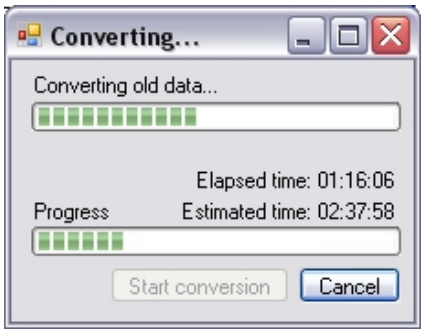
V této části se budu věnovat všem podpůrným úpravám a nástrojům, které jsem vytvořil pro správnou funkčnost ukládání grafových dat jako celku.

4.4.1 gcTester

První úpravou je zásah do zdrojového kódu programu *gcTester.exe*, který tvoří testovací jádro povádějící testy na základě dat v databázi. Zde došlo ke změně pouze několika řádků, které zajišťují správné uložení přímých grafových dat do databáze a pravidelné volání konverzí do posloupností.

4.4.2 DatabaseConverter

DatabaseConverter je nástroj, který jsem vytvořil pro konverzi starých dat z tabulky *GraphData_5m* na data nová do tabulky *GraphData*. Bude použit vždy pouze jednou pro konverzi starých klientských či centrálních dat. Po spuštění aplikace je uživatel vyzván, aby vložil údaje pro připojení do databáze. Následně může kliknutím spustit konverzi, která je složena ze dvou částí - konverze a testování správnosti dat. Stav konverze je zobrazován ve dvou indikátorech průběhu. Jeden ukazuje průběh aktuálního kroku, druhý pak celkový průběh. Zároveň je zobrazován uplynulý čas a odhad času zbývajícího. Obrázek 12 ukazuje okno s průběhem konverze.



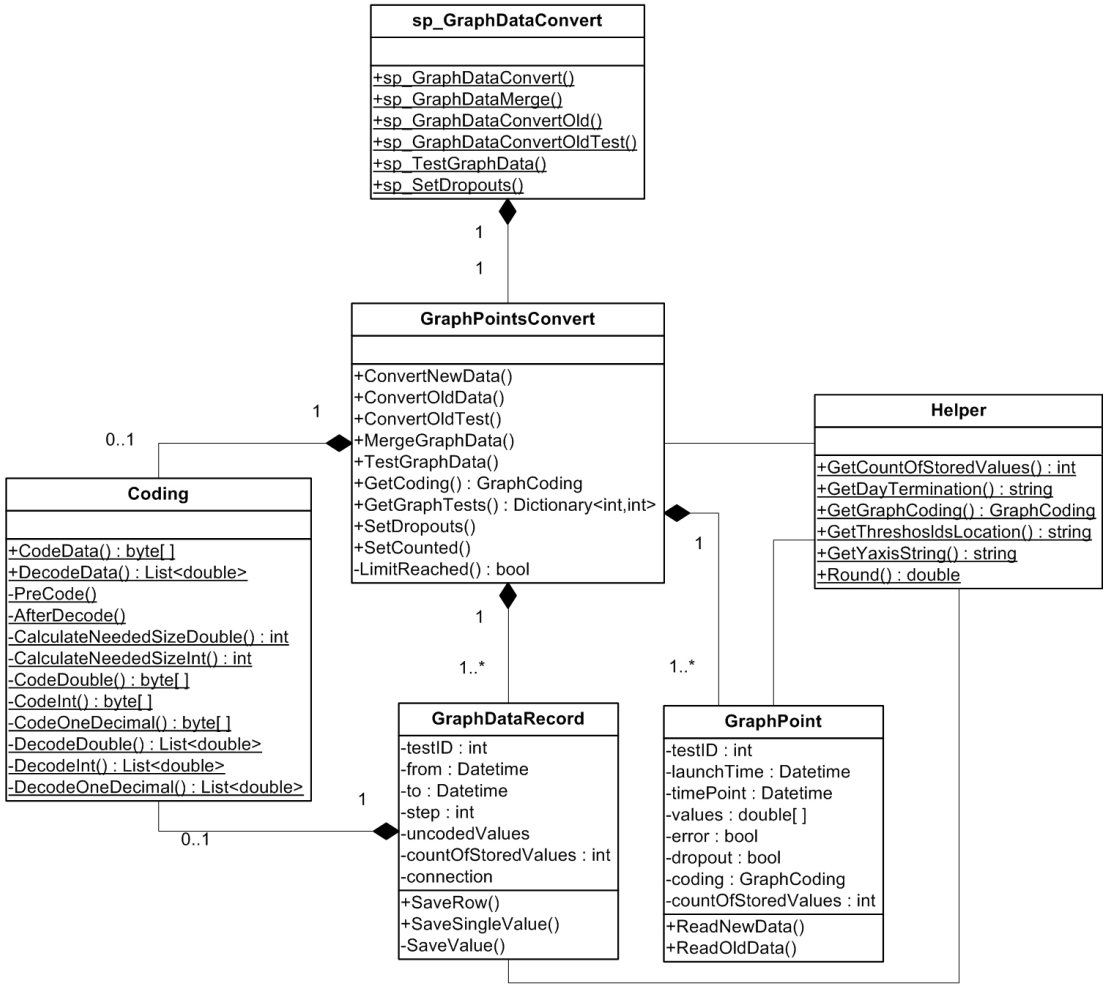
Obrázek 12: Probíhající konverze starých dat

4.4.3 GraphDataConvert.dll

GraphDataConvert.dll je knihovna vytvořená pro pravidelnou konverzi dat na databázovém serveru. Metody v ní obsažené jsou volané přímo jako uložené procedury. Obsahuje metody pro konverzi nových dat a pro spojování posloupností dohromady. Jsou to tyto uložené prodecury:

- **GraphDataConvert** - konverze nových nezpracovaných dat do nové posloupnosti.
- **GraphDataConvertOld** - konverze starých dat replikovaných od klientů. Jakmile budou mít všichni klienti novou verzi, nebude tato metoda potřebná.
- **GraphDataConvertOldTest** - konverze starých dat u jednoho konkrétního testu. Tato metoda byla vytvořena pro potřebu jednorázové konverze starých dat. Zpracovává pouze jeden test, aby se dal alespoň základním způsobem sledovat průběh konverze a odhadnout zbývající čas.
- **GraphDataMerge** - slouží pro spojení posloupností dohromady. Tento krok je oddělen od samotné konverze, aby se nemusely replikovat celé posloupnosti, ale pouze nově zpracované hodnoty.
- **TestGraphData** - testuje jeden konkrétní test. Přečte stará a nová data a nazvájem je srovná. Pokud se data nerovnají, je vyvolána výjimka *NotEqualDataException*.
- **SetDropouts** - tato metoda slouží k označování plánovaných výpadků. Je to pouze podpůrný prostředek, proto jej nebudu rozepisovat. Podrobnosti jsou uvedeny v programátorské příručce.

Třídní diagram knihovny je na obrázku 13. Uložená procedura vytvoří instanci třídy *GraphPointsConvert* a zavolá požadovanou metodu. Třída *GraphPoint* reprezentuje nezpracovaný naměřený výsledek testu a *GraphDataRecord* zastupuje nový řádek s posloupností zpracovaných hodnot. Třída *Coding* obsahuje metody pro kódování/dekódování dat. V normální aplikaci by byla implementována jako statická, SQL server však nedovoluje CLR uloženým procedurám ukládat data do statických proměnných. Proto musela



Obrázek 13: Třídní diagram knihovny *GraphDataConvert.dll*

být třída implementována jako instanční. *Helper* je pomocná třída obsahující metody pro získání informací o testech.

4.4.4 GuardControl.dll

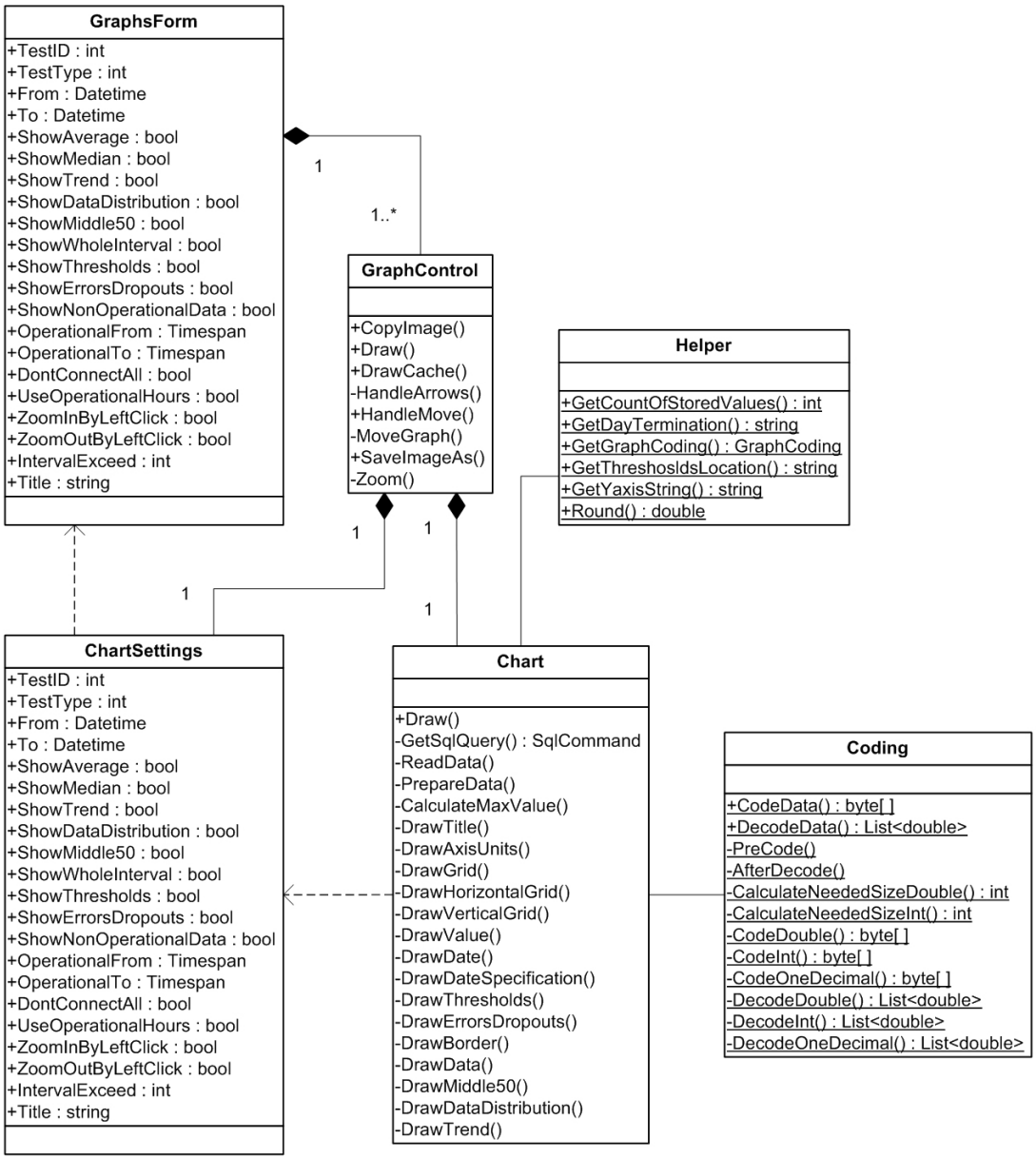
GuardControl.dll je zásuvný modul do aplikace *visionapp Application Delivery Management Suite 2008*. Tato knihovna poskytuje uživatelské rozhraní pro vytváření a správu testů. Došlo v ní ke změně třídy *GraphsForm*, odstranění staré logiky pro vykreslování a přidání logiky nové.

Třídní diagram grafů je znázorněn na obrázku 14. Pro přehlednost jsem neuváděl vstupní parametry metod a některé metody jsem vynechal (převážně metody reakcí na události, nebo metody nepodstatné pro pochopení problematiky). Třída *GraphsForm* obsahuje jednu či více (maximálně 4) instancí typu *GraphControl*. Každá instance *GraphControl* má právě jednu instanci třídy *Chart* a jednu instanci třídy *ChartSettings*. Třída *GraphsForm* pouze obsluhuje uživatelskou aktivitu, vystavuje vlastnosti pro nastavení grafu a vyvolává překreslení grafu. Třída *GraphControl* zajišťuje funkčnost, která není dostupná ve webové verzi. Vykresluje navigační šipky, zpracovává požadavky na uložení grafu do souboru či clipboardu a obsluhuje zoomování. O veškeré vykreslováni grafu se stará třída *Chart*. Informace o nastavení grafu získává z instance třídy *ChartSettings*, která je zprostředkování z hlavního formuláře. Tvoří tedy pouze prostředníka mezi těmito třídami. Mohlo by se zdát, že je tedy zbytečná a *Chart* si může informace brát přímo z hlavního formuláře. Pokud bych třídu *ChartSettings* vynechal, musel bych ve webové verzi měnit implementaci třídy *Chart* tak, aby správně fungovala. Díky *ChartSettings* to dělat nemusím. Třída *Chart* tedy zůstane pro obě verze stejná a změní se pouze použití třídy *ChartSettings*.

Při vykreslování jsou použity metody ze statických tříd *Helper* a *Coding*. *Helper* obsahuje několik metod pro získání statických informací o testech. Statická třída *Coding* obsahuje logiku pro kódování dat.

Současná verze se snaží základním způsobem cachovat data. Při výběru intervalu dojde k přečtení hodnot z databáze, které se následně zpracují. Vzniklé body grafu se uloží do paměti a vykreslí. Každé další vyvolání metody *OnPaint*, které může být způsobeno i pouhým přepnutím aktivní aplikace, způsobí vykreslení grafu ze zpracovaných bodů. Při změně požadovaného intervalu, nebo při zoomování se do databáze znovu odesílá požadavek na data, takže některá data se mohou číst z databáze vícekrát. Přenášení velkého objemu dat po síti zpomaluje reakci grafu a zatěžuje síť i databázový server.

V nové verzi dochází ke cachování dvojím způsobem. Po vykreslení dat je obrázek uložen v paměti. Kdykoli dojde k vyvolání metody *OnPaint*, ověří se, zda došlo ke změnám v nastavení a pokud ne, vykreslí se uložený obrázek z paměti. Toto opatření šetří výpočetní výkon zejména při zobrazení většího objemu dat; uživatel však nejspíše rozdíl nepozná.
Všechny hodnoty přečtené z databáze jsou uloženy ve slovníku typu *Dictionary<datetime, double>*. S hodnotami ve slovníku se nemanipuluje, pouze se z nich filtrují hodnoty pro zobrazení. Pokaždé, když uživatel jakkoli změní interval pro zobrazení, se nejprve ověří, zda požadovaná data již nejsou ve slovníku. Pokud ano, není je třeba číst znova z da-



Obrázek 14: Třídní diagram desktopové verze

tabáze. Pokud ne, z databáze se přečtou pouze ty hodnoty, které ještě ve slovníku nejsou. Díky tomu by nemělo docházet k situacím, kdy se z databáze některá data čtou vícekrát. To snižuje zatížení serveru a zrychluje práci s prostředím.

4.4.5 GC Monitor

GC Monitor je webová aplikace sloužící k rychlému přehledu stavu testů. Uživatel si zde může zobrazit celou strukturu svých serverů, nebo výsledky testů a grafy. Testy však vytvářet či modifikovat.

V této aplikaci jsem existující stránku vykreslující grafy upravil tak, aby ovládání bylo velmi podobné desktopové verzi. Pro implmentaci jsem použil technologii ASP.NET AJAX 1.0 a použil jsem několik komponent z balíku AJAX Control Toolkit. Díky tomu jsem mohl zachovat vzhled podobný desktopové verzi a díky asynchronním požadavkům na server je aplikace uživatelsky přívětivější

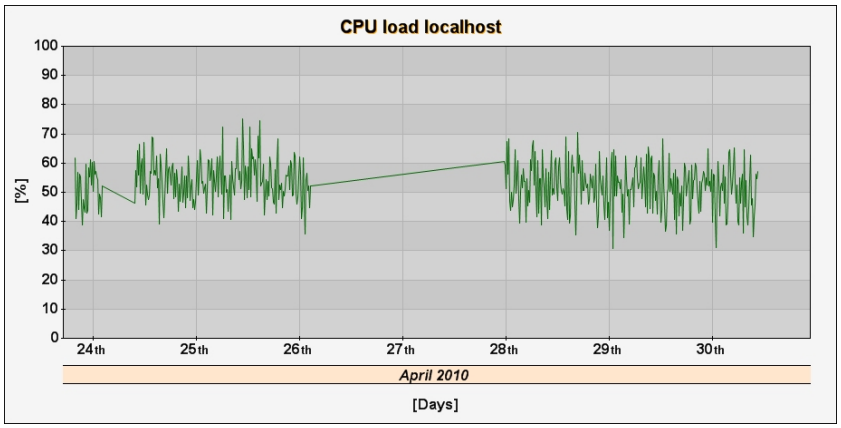
Ovládání grafu je umístěno na stránce *ShowGraph.aspx*. Graf je na stránku vložen jako obrázek s adresou *Graph.aspx*. Aby třída měla informace pro vykreslení grafu, ve třídě *ShowGraph.aspx.cs* se uloží nastavení do instance třídy *ChartSettings*, která se uloží do session. Odsud ji získá třída *Graph.aspx.cs*, která ji poskytne instanci třídy *Chart*.

4.4.6 Dostupné zdrojové kódy

Na přiloženém CD budou dostupné pouze zdrojové kódy, které jsem vytvořil. Většina z nich bohužel není spustitelná, protože zbývající zdrojové kódy aplikací nemohou být zveřejněny. Následuje seznam nástrojů, ke kterým popíšu zvěřejněný rozsah.

- **gcTester** - v testovacím jádru došlo pouze k drobným změnám v ukládání grafových dat. Zdrojové kódy nejsou dostupné.
- **GuardControl** - dostupné budou pouze kódy týkající se grafů, které jsem vytvořil. Z tohoto důvodu není možné si výsledek vyzkoušet.
- **GC Monitor** - na webovou verzi se vztahují stejná omezení jako na desktopovou.
- **GraphDataConvert** - ke knihovně jsou dostupné kompletně všechny zdrojové kódy.
- **DatabaseConverter** - aplikace pro konverzi dat je také plně dostupná.

Projekty *GraphDataConvert* a *DatabaseConverter* jsou součástí jedné solution, ke které patří i projekt *Tester*. Ten slouží pro debugování kódu, který normálně spouští databázový server jako uložené procedury. Projekt *Tester* je tedy pouze podpůrný nástroj, proto se o něm nebudu zmiňovat.



Obrázek 15: Zobrazení spojnice průměrů

4.5 Vykreslování dat

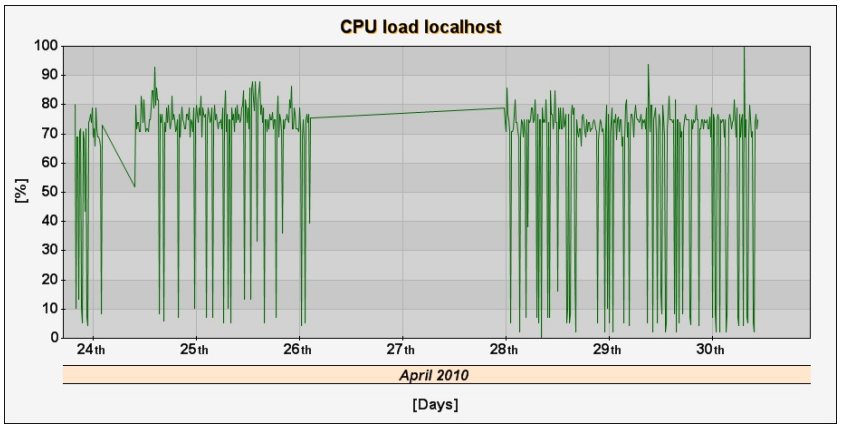
Oproti předchozí verzi, která vykreslovala pouze průměrnou hodnotu na základě zadaného časového intervalu, nová verze nabízí širší možnosti, díky kterým může uživatel získat více informací o datech z daného intervalu. Některé návrhy pro možnosti vykreslování vznikly po konzultaci s dalšími pracovníky firmy, kteří s grafy budou pracovat. V této sekci rozeberu podrobně jednotlivé možnosti zobrazení, včetně nastínění jejich implementace. Jednotlivé možnosti budu demonstrovat na ukázkách grafu vytížení procesoru osobního počítače.

4.5.1 Průměr

Tato volba zobrazuje na krátkém intervalu přímo naměřené hodnoty. V okamžiku, kdy na jeden pixel šířky grafu připadá více než jedna hodnota, vykresluje se průměr všech náležících hodnot. Průměr sice data zkresluje, což je problémem současné verze, ale jelikož je to velice srozumitelná charakteristika, nemá smysl ji opomíjet. Na obrázku 15 je průměrné vytížení procesoru za období dlouhé přibližně týden. Z grafu plyne, že průměrné vytížení procesoru bylo mezi 40 a 70 procenty. Tato informace sama o sobě příliš nevypovídá o tom, co se ve skutečnosti dělo. K tomuto obrázku budu vztahovat všechny následující ukázky.

4.5.2 Medián

Medián [17] se na krátkém intervalu chová stejně jako průměr. Když je překročena hranice jedné hodnoty na pixel, vykresluje se místo průměru medián. Ten se získá setříděním hodnot v pixelu podle velikosti a vybráním prostřední hodnoty. Pokud je počet hodnot sudý, vybere se průměr prostředních dvou hodnot. Obrázek 16 ukazuje stejná data jako v minulé ukázce, tentokrát však místo průměru zobrazuje medián. Graf budí dojem, že hodnoty jsou na intervalu více rozložené a že spíše tíhnou k hodnotě kolem 80%. Máme



Obrázek 16: Zobrazení spojnice mediánů

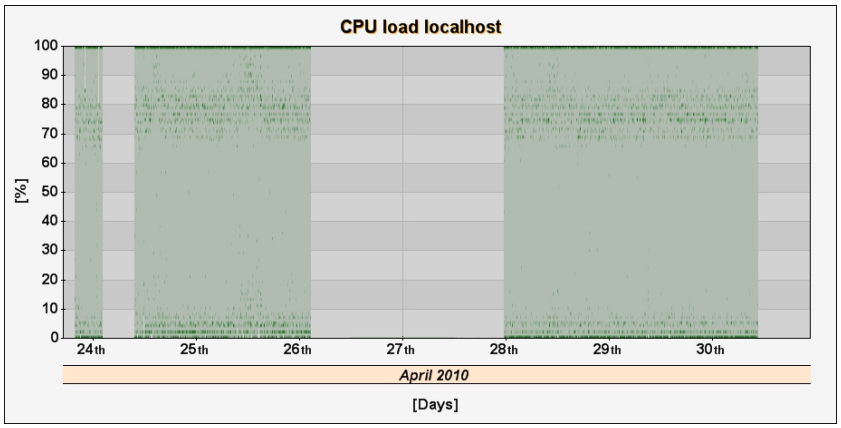
tedy informaci, kterou nám samotný průměr nedokázal poskytnout. Nicméně stále nemáme ponětí, jaké chování tento výsledek způsobuje. Obecně pro srovnání mediánu s průměrem platí, že čím jsou si oba průběhy podobnější, tím lépe vypovídají o skutečné situaci.

4.5.3 Rozložení dat

Pomocí této vrstvy se dá získat komplexnější pohled na data v daném intervalu. Vyplatí se ho používat spíše ve vzdálenějších pohledech. Celý interval, ve kterém se data nachází, je vybarven základní světlou barvou a ostatní odstíny jsou tvořeny výskytem hodnot v konkrétní lokalitě. Tmavší oblast znamená vyšší zastoupení hodnot.

Po implementační stránce je nejprve vytořeno dvojrozměrné pole hodnot, které reprezentují jednotlivé sloupce a řádky grafu. Pro každý sloupec grafu se projdou všechny hodnoty, které mu náleží a zaznamenají se do pole přičtením hodnoty v daném místě a okolí. Při vykreslování se nejprve spojí minimální a maximální prvek v každém sloupci nejsvětlejším odstínem. Nejvyšší hodnotě v poli je přiřazena maximální intenzita barvy a pro ostatní se hodnota spočítá relativně k té největší. Čím větší je počet hodnot umístěných v pixelu, nebo jeho těsné blízkosti, tím větší číslo v poli bude pixel mít, takže bude vykreslen tmavším odstínem. Výsledek tedy zobrazuje relativní zastoupení naměřených hodnot v jednotlivých sloupcích. Toto zobrazení může zkreslovat ve chvíli, kdy není v celém grafu stejná hustota dat, protože v každém sloupci se tvoří barva relativně k počtu hodnot v daném sloupci.

Ukázka rozložení dat pro týdenní interval vytížení procesoru je na obrázku 17. Nyní máme velmi dobrou představu o tom, co se na počítači ve skutečnosti dělo. Vidíme, že hodnoty se nachází především ve třech oblastech - mezi 0 a 10 procenty, dále mezi 70 a 90 procenty a mnohokrát byl také procesor vytížen na 100%. Je tedy vidět, že naměřené hodnoty velmi kolísaly, což bychom například z průměru nevyčetli, dokud bychom si graf nepřiblížili. Rozložení dat tedy poskytuje komplexnější informaci, než pouhý průměr či medián.



Obrázek 17: Zobrazení rozložení dat

4.5.4 Prostředních 50%

Touto vrstvou je zobrazeno prostředních 50% hodnot ze setříděného seznamu. Tato charakteristika bývá ve statistice označována jako interkvartilové rozpětí [18]. Narozdíl od průměru není zobrazení ovlivněno krátkodobými špičkami, protože filtruje shora i zdola čtvrtinu dat. Pro krátké intervaly se nemusí zobrazovat z důvodu nedostatku dat. Aby byla tato vrstva vykreslena, musí náležet jednomu pixelu alespoň čtyři hodnoty.

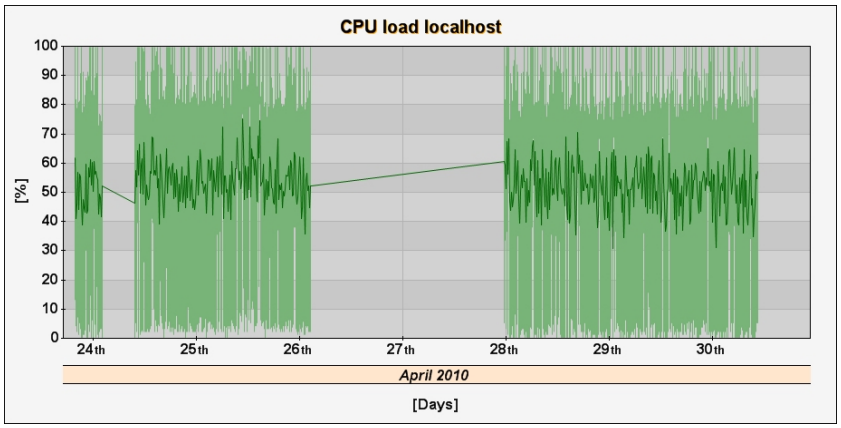
Na našem příkladě kolísající zátěže procesoru na obrázku 18 tato vrstva vybarví postatnou část grafu, což nám opět dává jasný signál o tom, jak hodnoty kolísají. Z velikosti intervalu prostředních 50% a jeho vzájemné polohy s průměrem se dají vyčíst tyto informace:

- Čím je interval prostředních 50% menší, tím menší rozptyl mají hodnoty, které se v něm nacházejí
- Pokud průměr prochází zhruba uprostřed intervalu, pak čím je interval kratší, tím má průměr větší váhu.
- Pokud průměr prochází krajem intervalu, zmanená to, že se v jedné z odfiltrovaných oblastí nachází špičky, které průměr táhnou svým směrem.
- Může se také stát, že průměr neprochází intervalem prostředních 50%. To může nastat, když se ve sloupci vyskytuje malý počet hodnot, které se velmi výrazně liší od hodnot ostatních.

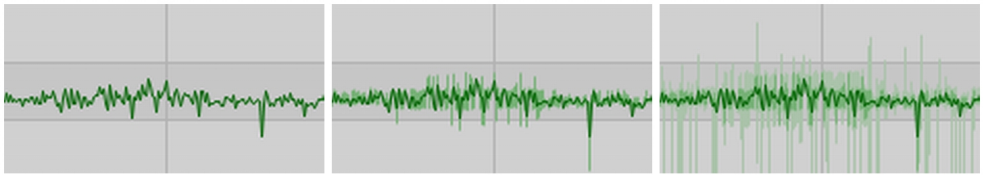
V našem příkladě velký interval prostředních 50% dat správně poukazuje na kolísání hodnot.

4.5.5 Celý interval

Zobrazení celého intervalu v každém sloupci grafu spojuje nejmenší a největší hodnotu. Tato vrstva stejně jako prostedních 50% upřesňuje pohled na data. Zachovává všechny



Obrázek 18: Zobrazení prostředních 50% dat



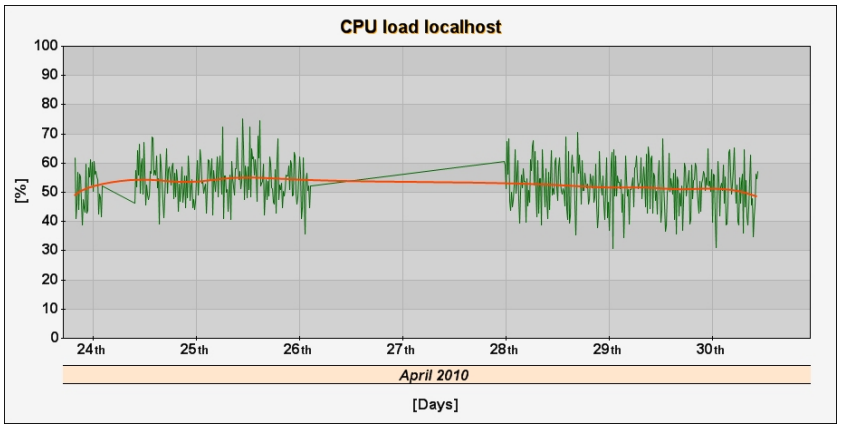
Obrázek 19: Přidávání vrstev při pohledu na stejná data

krátkodobé špičky, které nemusí být patrné z průměru, nebo prostředních 50%. Příklad procesoru, na kterém demostruji jednotlivé vrstvy, by tuto vrstvu vykreslil jako plnou oblast shora dolů, proto ukážu rozdíl na jiném příkladě. Na obrázku 19 jsou zobrazeny tři pohledy na stejný průběh. První pohled (a) zobrazuje pouze průměr. Druhý (b) ukazuje stav, kdy se k průměru přidá vrstva prostředních 50%. V posledním pohledu (c) je k těmto dvěma přidán celý interval. Dá se tedy říci, že měřená charakteristika vykazovala stabilní chování s pouze malými lokálními výkyvy.

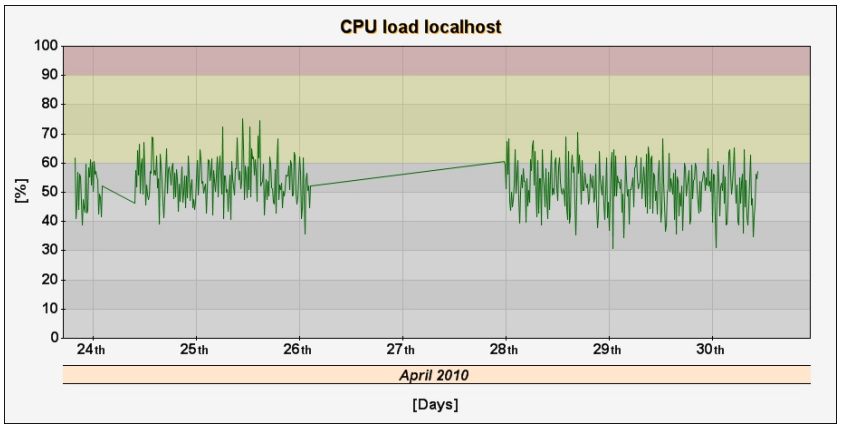
4.5.6 Trend

Trend ukazuje nad daty aproximační křivku. Ze zobrazených dat se nejprve vypočítají průměry hodnot po deseti pixelech. Do průměrů se počítají pouze sloupce, ve kterých se nachází hodnoty. Vznikne pole průměrů, které tvoří vodící body Bézierovy křivky [19]. Spočítá se 100 bodů, které tvoří křivku a ta je vykreslena. Výsledná křivka začíná a končí v krajních bodech, ostatní ji pouze ovlivňují.

Na příkladu se zátěží procesoru jde z obrázku 20 vidět, že trend nestoupá ani neklesá. Trend tedy poskytuje jakousi souhrnnou informaci o datech, kterou si lze většinou celkem jednoduše domyslet. Může však nastat situace, kdy bude v několika po sobě jdoucích sloupcích jen málo hodnot, které budou vysoké a ve zbytku sloupců bude velké množství rozmanitých hodnot. Těchto několik vysokých hodnot bude mít vysoký průměr,



Obrázek 20: Zobrazení trendu

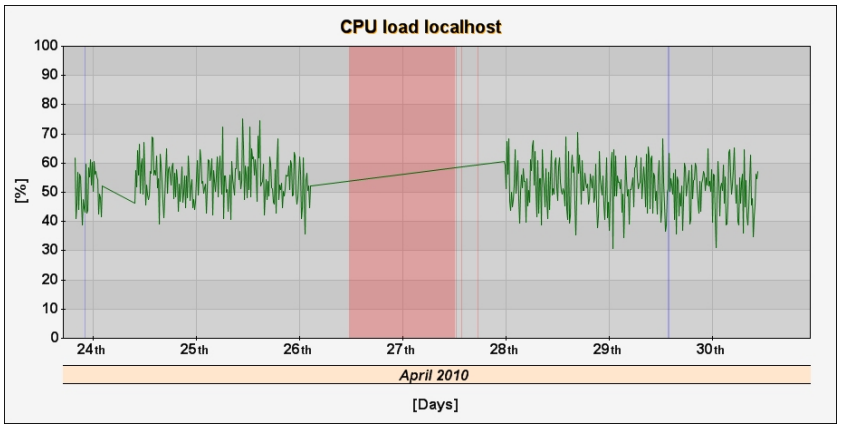


Obrázek 21: Zobrazení hraničních hodnot

zatímco zbytek grafu bude mít průměr nižší. V takovéto situaci by křivka trendu nebyla výrazně ovlivněna malou oblastí vysokých hodnot, což naznačuje, že oblast s vysokým průměrem nemá velký význam vzhledem k celku.

4.5.7 Hraniční hodnoty

U vybraných testů lze nastavit dvě hraniční hodnoty (žlutou a červenou), které neslouží pouze k zobrazování, ale i k reakcím ve formě notifikací či předem definovaných opatření. Slouží také k rozlišení stavu testu. V grafu jsou obě hraniční hodnoty zobrazeny vybarvením plochy, která jim náleží. Pozadí je překlesleno částečně průhlednými odstíny žluté a červené barvy. Pokud se vrstva nezobrazuje, znamená to, že se nachází mimo zobrazený interval dat, nebo že test hraniční hodnoty neukládá. Na obrázku 21 jsou zobrazeny hraniční hodnoty 60 a 90% pro test zátěže procesoru.



Obrázek 22: Zobrazení chyb a plánovaných výpadků

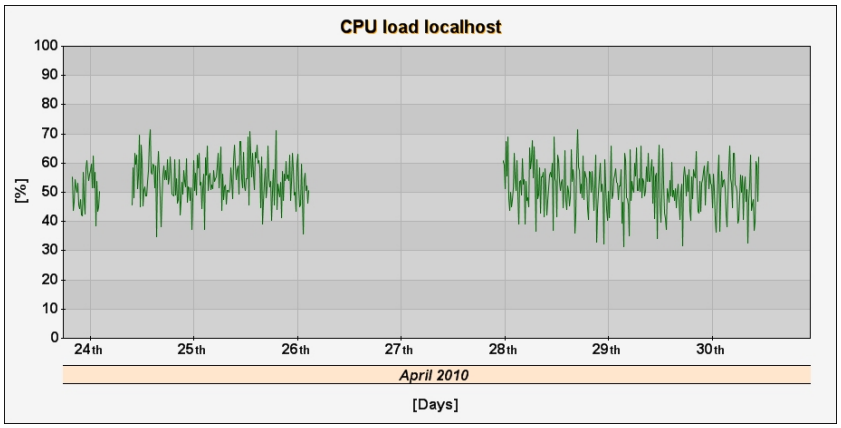
4.5.8 Chyby a plánované výpadky

Chyby a plánované výpadky jsou zobrazovány v místě svého výskytu jako svislá čára přes celý graf. Chyby mají červenou barvu, plánované výpadky modrou. Pokud se v pixelu nachází chyba i výpadek, bude vykreslena chyba, protože má vyšší důležitost. Vrstva je vidět na obrázku 22. Na grafu je červená oblast značící velké množství chyb. Jsou vidět i dva plánované výpadky, které však neměly dlouhé trvání.

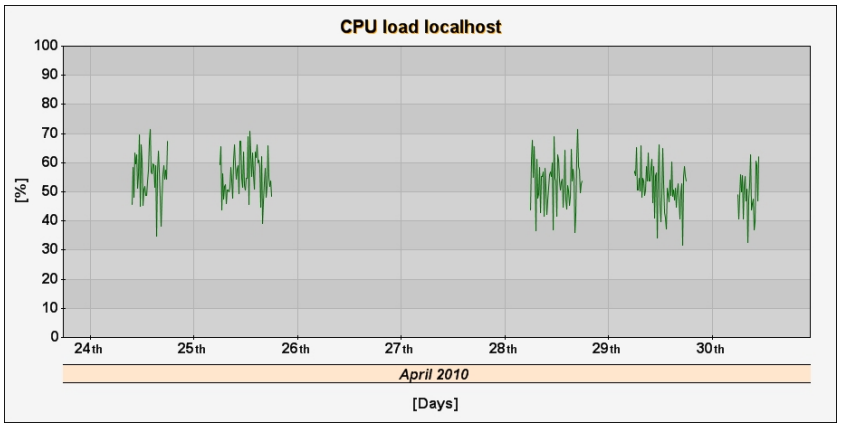
4.5.9 Filtrování dat

Na příkladech, které předcházely, byl hned na dvou místech k vidění jeden nežádoucí jev - ve chvíli, kdy chybí část dat (počítač byl třeba vypnutý), jsou předešlá a následující data spojena čarou, která očividně nereprezentuje žádný skutečný vývoj hodnot. Jednoduchým zadáním maximální doby pro spojování hodnot lze nežádoucí spojnice vyfiltrovat. Víme-li například, že test je spouštěn pravidelně jednou za minutu, můžeme limit nastavit třeba na deset minut. Jakýkoli interval mezi dvěma hodnotami, který přesahuje limit, nebude spojen. Výsledek je na obrázku 23.

Druhým způsobem jak filtrovat data je nastavení provozních hodin, pro které má uživatel dvě možnosti ovlivnění výsledku. Buďto si zvolí, aby se hodnoty mimo interval provozních hodin nezobrazovaly vůbec, nebo se tyto hodnoty nebudou započítávát do zobrazovaných průměrů. Druhý způsob se tedy projeví pouze na vzdálenějších pohledech, ale ani tak nezpůsobuje výraznou změnu v grafu. Zajímavější volbou je proto nezobrazovat hodnoty mimo interval vůbec, jak je ukázáno na obrázku 24, kde jsou provozní hodiny nastaveny na 6:00 - 18:00. Na dlouhodobém pohledu se filtrování vyplatí pro sledování vývoje jevu, který nastává v určitou dobu.



Obrázek 23: Filtr spojování intervalů přesahujících limit



Obrázek 24: Filtr neprovozních hodin

4.5.10 Priority vrstev

Vrsty jsou vykreslovány ve stanoveném pořadí, které vytváří jakési priority vrstev. V následujícím seznamu jsou postupně seřazeny vrstvy počínaje nejvyšší prioritou - tedy vrstvou, která bude vždy nahoře. Pokud je na řádku více hodnot, znamená to, že může být vykreslena pouze jedna z uvedených vrstev.

1. Trend
2. Průměr, Medián, Rozložení dat
3. Prostředních 50%, Rozložení dat
4. Celý interval, Rozložení dat
5. Chyby a plánované výpadky
6. Hraniční hodnoty

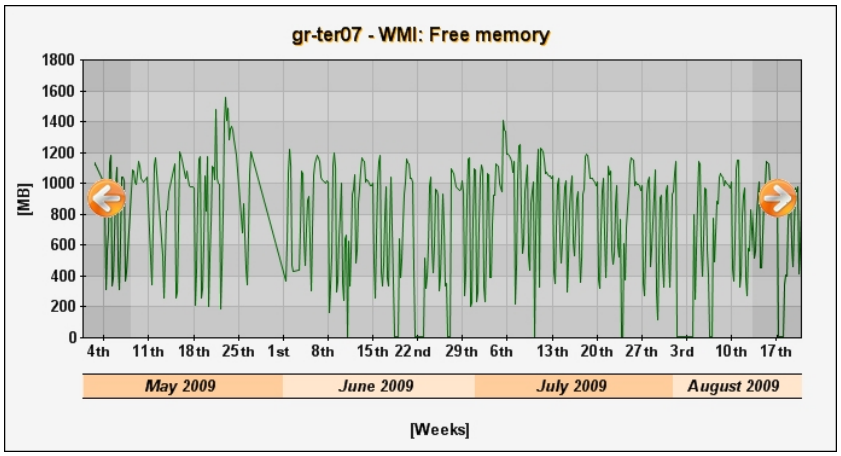
4.6 Okolní vzhled

Okolí grafu zaznamenalo pouze jednu novinku - přibyla druhá časová osa, která upřesňuje datum zobrazený na ose x. Udržet v grafu vazbu na místo v čase není snadný úkol. Ve vzdáleném pohledu není problém popsat časovou osu datem ve formátu MM/y-yyy. Problém nastane ve chvíli, kdy graf zobrazuje období kratší, například několik hodin. Pokud osu popíšeme pouze časem, není z grafu jasné, který den je zobrazen. Uživatel sice ví, kde se nachází, protože s grafem pracuje, ale pokud by byl graf prezentován třetí straně, přesné vymezení v čase bude chybět. Pokud zvolíme dlouhý formát data (d.M.yyyy hh:mm:ss), máme sice kompletní informaci o čase, popisek je ovšem tak dlouhý, že se na jich na osu vejde jen několik. Zkoušel jsem popisky natočit o 45°, ale výsledek nebyl uspokojivý. Teprve díky druhé časové ose je nyní v každém okamžiku dostupná úplná informace o místě v čase.
Kromě druhé časové osy zůstalo rozvržení okolí grafu stejné jako v minulé verzi.

4.7 Rozdíly mezi desktopovou a webovou verzí

Obě verze skýtají stejné možnosti vykreslování dat, dá se tedy pomocí obou dosáhnout stejného výsledku. Desktopová verze má vykreslený graf čistější, protože nedochází ke kompresi do formátu jpeg. Mezi desktopovou a webovou aplikací jsou tyto rozdíly:

- Prostřednictvím desktopové aplikace si uživatel může graf přímo uložit do souboru nebo do clipboardu.
- Desktopová aplikace podporuje ukládání uživatelských nastavení.
- Ve webové verzi se dá zoomovat pouze pomocí dvou tlačítek s lupou. Přiblížení nebo oddálení se vztahuje ke středu grafu.



Obrázek 25: Umístění oblastí navigačních šipek

- Desktopová aplikace podporuje několik možností zoomování. Uživatel může použít pro přiblížení/oddálení tyto dvojice akcí: (levý dvojklik/pravý klik), (scroll nahoru/scroll dolů), (v horním menu označená lupa +/- a levý klik).
- V desktopové verzi se dá posunovat grafem doleva a doprava pomocí šipek, které se zobrazí u okraje grafu při najetí myší do této oblasti. Kliknutím na šipku se graf posune o čtvrtinu intervalu. Na obrázku 25 jsou znázoněny oblasti pro zobrazení navigačních šipek.

4.8 Přechod ze staré na novou verzi

Přechod na novou verzi proběhne během provozu aniž by se přerušoval chod aplikací na delší dobu, než je nutná pro nakopírování nové verze. Přechod bude rozdělen do dvou fází, z nichž každá bude mít jako výsledek funkční celek.

4.8.1 Přechod centrální databáze

Prvním krokem bude přechod na novou verzi v centrální databázi. Nejprve se do databáze přidají nové tabulky a funkčnost. Provede se konverze sratých dat na nové, nové pětiminutové průměry, které budou replikovány od klientů, se budou konvertovat do posloupností. Na webový server se doinstalují rozšíření pro AJAX a nasadí se nová verze aplikace. Stará data zatím nebude možno odstranit.

Po prvním kroku tedy budou v databázi obě verze grafových dat, nová data se budou konvertovat ze staré reprezentace do nové, odkud budou k dispozici všem klientům prostřednictvím webové aplikace.

4.8.2 Přechod u klientů

Ve druhé fázi se provede přechod na novou verzi u klientů. Do databáze se přidají tabulky a funkčnost, provede se konverze starých dat, nakopírují se nové verze aplikací a odstraní se stará data. Od té chvíle bude mít klient kompletní novou verzi.

Jakmile budou mít novou verzi všichni klienti, můžou se z databáze odstranit stará data a funkce, které je obsluhují.

5 Výsledek

V této kapitole se zaměřím na zhodnocení dosažených výsledků.

5.1 Nová reprezentace starých dat

Tato část obsahuje srovnání starých dat reprezentovaných starým způsobem a po konverzi na nový způsob. Vzorek databáze pochází z období 24.4.2009 - 26.8.2009 (124 dní, tedy přibližně třetina roku) z ostrého provozu. Srovnání velikostí tabulek GraphData_5m a GraphData je v tabulce 7. Rapidní úbytek potřebného prostoru a řádků, který je z tabulek vidět, je způsoben především průměrováním dat po pěti minutách. Nezávisle na tom, co se během těch pěti minut stalo, se vytvoří průměry, které na sebe velmi dobře navazují. Proto není velká potřeba vytvářet nové řádky, čímž se šetří místo. V nové verzi, kde se budou ukládat všechny hodnoty, pravděpodobně bude třeba nové řádky vytvářet častěji. Srovnání nové podoby dat s celkovými nároky na uložení pomocí aktuální verze je v tabulce 8, kde jsou započítány i tabulky se vzdálenějšími průměry.

5.2 Časová náročnost přístupu k datům

Novou a starou reprezentaci dat jsem sorvnával i z hlediska rychlosti přístupu k datům. Testování proběhlo na osobním počítači, na kterém je nainstalované uživatelské rozhraní pro zobrazování grafů i databázový server. Pro testování jsem vybral 20 různě velkých časových intervalů, které jsem nechal zobrazit pomocí obou způsobů. Před každým měřením jsem vymazal vyrovnávací paměť SQL serveru. Průměrná hodnota všech dvaceti vykreslení dat je pro starý způsob 154 ms, pro nový 97 ms. Tento rozdíl není zásadní, dá se však očekávat, že ve chvíli, kdy vykreslovací logika nebude umístěna na stejném stroji jako databázový server, prohloubí se tento rozdíl kvůli síťovému provozu.

5.3 Nová reprezentace nových dat

V této části srovnávám jakých výsledků dosáhne verze nová oproti verzi současné. Data pocházejí z období 7.4.2010 - 30.4.2010 (23 dní) a byla nasbírána na osobním počítači, na kterém byly v souběžném provozu obě verze ukládání dat. Výsledky jsou v tabulce 9. Výsledky jsou pouze orientační, protože nejsou dostatečně reprezentativní hned z několika důvodů:

- Měřený interval není příliš dlouhý.
- Rozložení jednotlivých testů neodpovídá rozložení testů na serverech.
- Servery pracují nepřetržitě - testovaný počítač byl často přepínán do úsporného režimu, či vypínán úplně.

Dá se tedy předpokládat, že v reálném provozu na serverech by byl počet řádků nižší, a to díky menší nutnosti vytvářet nové posloupnosti. Velikost dat bude závislá především na periodě mezi testy.

	GraphData_5m	GraphData	Procent
Počet uložených hodnot	91 797 946	87 579 086	95,40%
Počet řádků	91 797 946	473 979	0,52%
Velikost dat [kB]	9 712 832	107 152	1,10%
Velikost indexů [kB]	86 256	744	0,86%

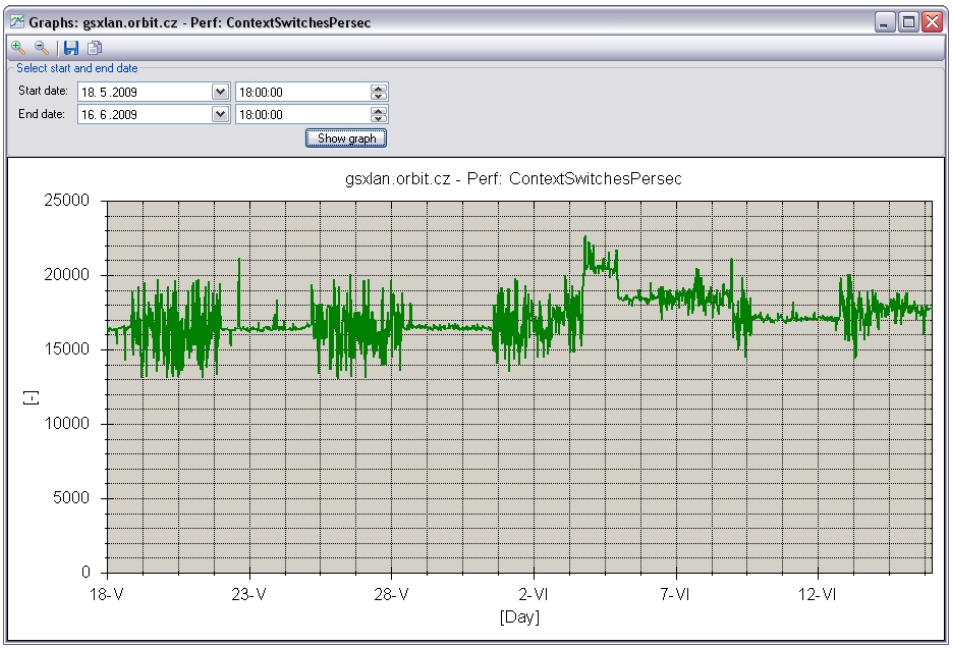
Tabulka 7: Srovnání tabulek GraphData_5m a GraphData

	5m + 30m + 2h + 12h	GraphData	Procent
Počet uložených hodnot	127 523 430	87 579 086	68,68%
Počet řádků	127 523 430	473 979	0,37%
Velikost dat [kB]	13 565 864	107 152	0,79%
Velikost indexů [kB]	105 856	744	0,70%

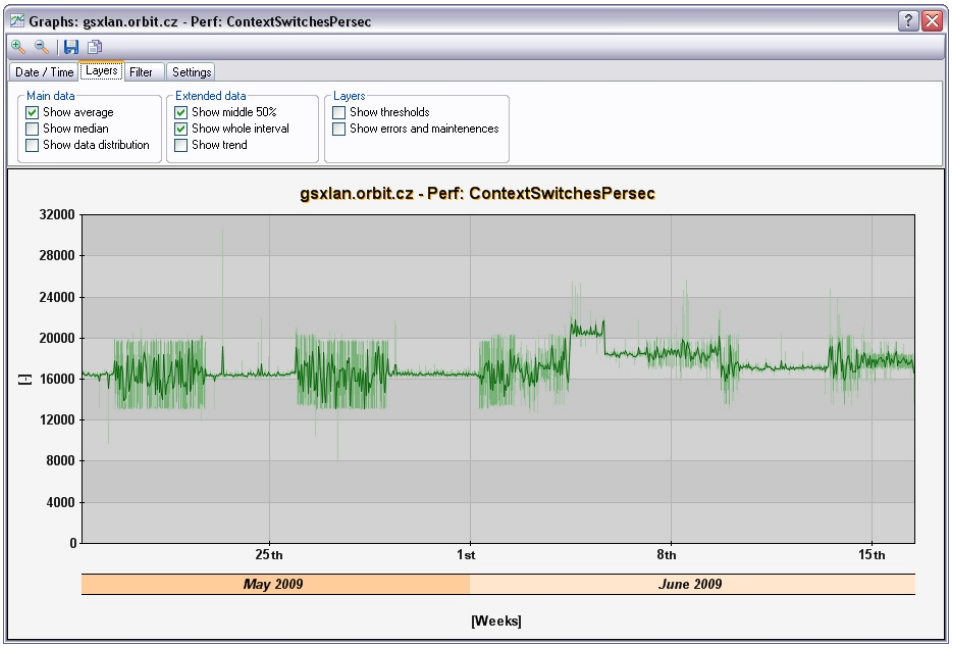
Tabulka 8: Srovnání celkových nároků na uložení grafových dat

5.4 Vzhled grafů

Nová verze poskytuje nejen přívětivější vzhled, ale i nové možnosti zobrazování dat. K dispozici jsou dvě verze - desktopová a webová. Pomocí obou se dá dosáhnout stejným způsobem stejného výsledku. Srovnání vzhledu staré verze a obou nových verzí je na obrázcích 26, 27 a 28.



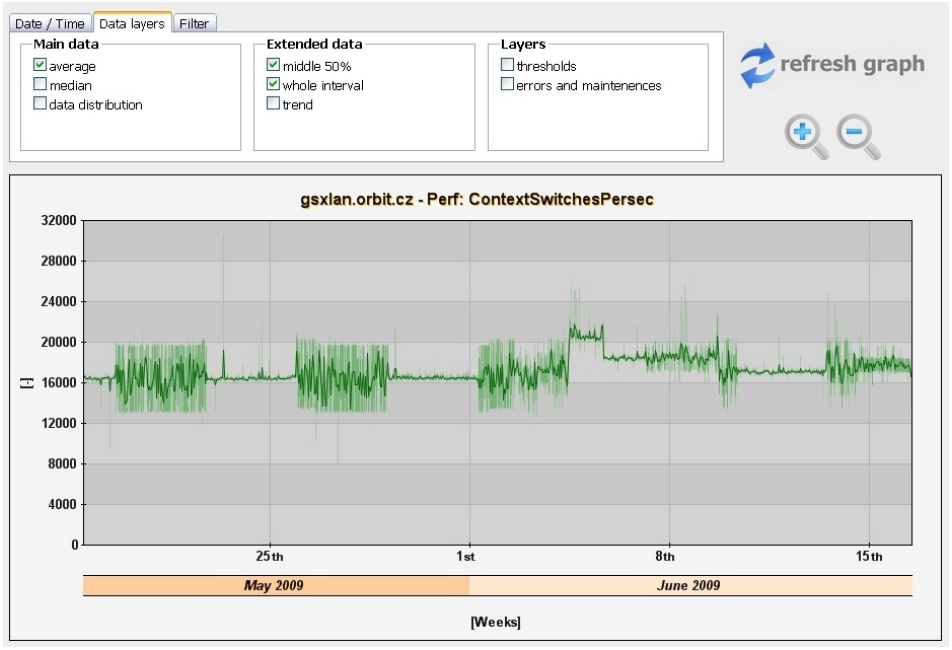
Obrázek 26: Vzhled předchozí verze



Obrázek 27: Vzhled desktopové verze

	GraphData_5m	GraphData	Procent
Počet uložených hodnot	71 333	335 823	470,78%
Počet řádků	71 333	3 749	5,26%
Velikost dat [kB]	7 808	818	10,48%
Velikost indexů [kB]	48	16	33,33%

Tabulka 9: Srovnání nových dat uložených pomocí GraphData_5m a GraphData



Obrázek 28: Vzhled webové verze

6 Závěr

Cílem práce bylo snížit zatížení databázového serveru způsobené velkým objemem ukládaných grafových dat a zároveň zlepšit grafický výstup těchto grafů. Z dosažených výsledků plyne, že stanovených cílů bylo dosaženo, v některých případech dokonce výsledky předčily mé vlastní očekávání.

Námětem pro další vývoj by mohly být především další pokročilé grafické a statistické výstupy. Bylo by zajímavé mít možnost navzájem srovnat výsledky testů napříč různými servery, nebo srovnat dva různé testy a zjistit, do jaké míry jsou jejich výsledky na sobě závislé.

Díky této práci jsem nahlédl do fungování firmy a poznal některé její potřeby a problémy. Práce mi také poskytla pohled do architektury reálného systému, který pracuje s velkým objemem dat a používají ho desítky firem.

Doufám, že po nasazení, kterého se budu částečně účastnit, bude výsledek mojí práce ve firmě ORBIT s.r.o. splňovat svůj účel a ulehčovat práci všem, kteří se s ním setkají.

7 Reference

[1] *ORBIT GuardControl - Monitoring ICT*
http://www.monitoringit.cz/

[2] *Extensible Markup Language*
http://cs.wikipedia.org/wiki/Extensible_Markup_Language

[3] *Estimating the Size of a Table*
http://msdn.microsoft.com/en-us/library/aa933068(SQL.80).aspx

[4] *bit (Transact-SQL)*
http://msdn.microsoft.com/en-us/library/ms177603.aspx

[5] *Fibonacci coding*
http://en.wikipedia.org/wiki/Fibonacci_coding

[6] *Fibonacciho posloupnost*
http://cs.wikipedia.org/wiki/Fibonacciho_posloupnost

[7] *Delta encoding*
http://en.wikipedia.org/wiki/Delta_encoding

[8] *ASP.NET*
http://en.wikipedia.org/wiki/ASP.NET

[9] *C# 2.0*
http://cs.wikipedia.org/wiki/C_Sharp#C.23_2.0

[10] *SQL Server 2005*
http://www.microsoft.com/cze/windowsserversystem/sql/default.mspx

[11] *Visual Studio 2005*
http://msdn.microsoft.com/en-us/library/ms950416.aspx

[12] *ASP.NET AJAX 1.0*
http://www.microsoft.com/downloads/details.aspx?FamilyID=ca9d90fa-e8c9-42e3-aa19-08e2c027f5d6&displaylang=en

[13] *Ajax Control Toolkit*
http://www.asp.net/ajaxlibrary/act.ashx

[14] *Database Objects*
http://msdn.microsoft.com/en-us/library/ms172451(SQL.90).aspx

[15] *binary and varbinary (Transact-SQL)*
http://msdn.microsoft.com/en-us/library/ms188362.aspx

[16] *Shrink Database Task*
http://msdn.microsoft.com/en-us/library/ms141819.aspx

[17] *Medián*
http://cs.wikipedia.org/wiki/Medi%C3%A1n

[18] *Interquartile range*
http://en.wikipedia.org/wiki/Interquartile_range

[19] *Bézierova křivka*
http://cs.wikipedia.org/wiki/B%C3%A9zierova_k%C5%99ivka

A Obsah přiloženého CD

Na přiloženém CD můžete najít tuto adresářovou strukturu:

A.1 Programátorská příručka

Programátorská příručka obsahuje soubor *index.html*, který stačí spustit a dále se příručka ovládá z prostředí prohlížeče. Dále obsahuje adresář *soubory*, ve kterém jsou další HTML soubory příručky, soubor s CSS stylem a javascriptový soubor obsahující menu a funkce pro jeho zobrazování.

A.2 Text

Tento adresář obsahuje zdrojové kódy tohoto textu, včetně použitých obrázků.

A.3 Zdrojové kódy

Zde jsou obsaženy adresáře se zdrojovými kódy, které jsem vytvořil v rámci této práce:

- *GC Monitor* - soubory k webové verzi, samostatně nespustitelné.
- *GraphDataConvert* - soubory týkající se konverze, sice spustitelné, z důvodu absence databáze však nevyzkoušitelné.
- *GuardControl* - soubory k desktopové verzi, samostatně nespustitelné.

A.4 SQL skripty

Adresář *SQL skripty* obsahuje 4 skripty:

- *DatabaseChanges.sql* - obsahuje změny v databázi pro novou verzi
- *DatabaseChangesRollback.sql* - ostraňuje změny
- *DeleteOldData.sql* - maže stará data z průměrovaných tabulek
- *DatabaseStatistics.sql* - zobrazuje statistiku nových a starých dat (počty řádků, objemy dat a podobně)

B Manuál pro používání grafů v GuardControlu

Ovládací část okna se zobrazeným grafem je rozdělena do těchto záložek:

B.1 záložka Date / Time

V této části se nastavuje datum a čas intervalu, který se má zobrazit. Maximální zobrazitelný interval je nastaven na dva roky, minimální pak na půl hodiny.

B.2 záložka Layers

Jak již název napovídá, jedná se zobrazování vrstev v grafu. Následuje rozpis jednotlivých vrstev.

average
Tato vrstva zobrazuje buď spojnici naměřených hodnot, nebo spojnici jejich průměrů, pokud je zobrazen delší interval. Nemůže být zobrazena zároveň s vrstvami *median* a *data distribution*.

median
Medián je prvek, který se nachází uprostřed setříděného seznamu dat. Oproti průměru není ovlivněn extrémními hodnotami. Pokud je interval dostatečně krátký, má tato vrstva stejný výsledek jako vrstva *average*. Vrstva nemůže být zobrazena s vrtvami *average* a *data distribution*.

data distribution
Pomocí této vrstvy se dá získat komplexnější pohled na data v daném intervalu. Vyplatí se používat spíše ve vzdálenějších pohledech. Celý interval, ve kterém se data nachází, je vybarven základní světlou barvou a ostatní odstíny jsou tvořeny výskytem hodnot v konkrétní lokalitě. Tmavší oblast znamená vyšší zastoupení hodnot. Zároveň s touto vrstvou nelze zobrazit vrstvy *average*, *median*, *middle 50%* a *whole interval*.

middle 50%
Touto vrstvou je zobrazeno prostředních 50% hodnot ze setříděného seznamu. Tato charakteristika bývá ve statistice označována jako interkvartilový rozptyl. Pro krátké intervaly se nemusí zobrazovat z důvodu nedostatku dat. S touto vrstvou nelze vykreslit vrstvu *data distribution*.

whole interval
Zobrazení celého intervalu v každém sloupci grafu spojuje nejmenší a nejvyšší hodnotu. Stejně jako prostředních 50% je tato vrstva nesloučitelná s vrstvou *data distribution*.

trend
Trend ukazuje aproximační křivku, která aproximuje průměry aktuálně zobrazeného

intervalu. Není konfliktní s ostatními vrstvami

thresholds
Tato vrstva formou podbarvení pozadí zobrazuje hodnoty, které byly nastaveny jako hraniční (žlutá a červená hranice). Pokud oblasti nejsou vidět, znamená to, že jsou mimo rozsah zobrazovaný grafem. Spolu s touto vrstvou je možno zobrazit jakékoli jiné vrstvy.

errors and maintenances
Chyby a plánované výpadky jsou vykreslovány svislou čarou v místě výskytu. Chyby mají světle červenou barvu, plánované výpadky barvu modrou. Tato vrstva je rovněž zobrazitelná se všemi ostatními.

B.3 záložka Filter

Zobrazovaná data lze filtrovat dvěma způsoby:

- Nastavením provozních hodin lze dosáhnout jednoho ze dvou efektů - data mimo interval se buď nebudou zobrazovat vůbec, nebo se pouze nebudou započítávat do dlouhodobějších průměrů.
- Druhý způsob filtrování dat je nastavení délky intervalu, pro který se hodnoty spojují - pokud delší dobu nejsou dostupné výsledky testů, hodnoty jsou přesto spojeny čarou, která však nemá vypovídací schopnost. Právě pro tento účel byla přidána možnost nespojovat hodnoty, mezi kterými uplynul volitelný časový interval.

B.4 záložka Settings

V této části se nachází pouze jedna volba. Uživatel si může vybrat, jestli chce mít uloženo veškeré nastavení týkající se vrstev a fitrů. Tato možnost není dostupná ve webovém rozhraní.

B.5 Ovládání grafu

Graf lze ovládat těmito způsoby: (platí pouze pro desktopovou verzi)

- Dvojklik levým tlačítkem v grafu způsobí přiblížení pohledu v místě kliknutí.
- Pravý klik v grafu způsobí oddálení pohledu v místě kliknutí.
- Přiližovat a oddalovat pohled se dá také scrollovacím tlačítkem myši.
- Přiblížení či oddálení se dá provést i stisknutím tlačítka s lupou v horním menu, následně se akce provádí jedním kliknutím levého tlačítka myši.
- Při přiblížení myší k levému či pravému okraji grafu se zobrazí šipka. Po kliknutí dojde k posunutí grafu požadovaným směrem.

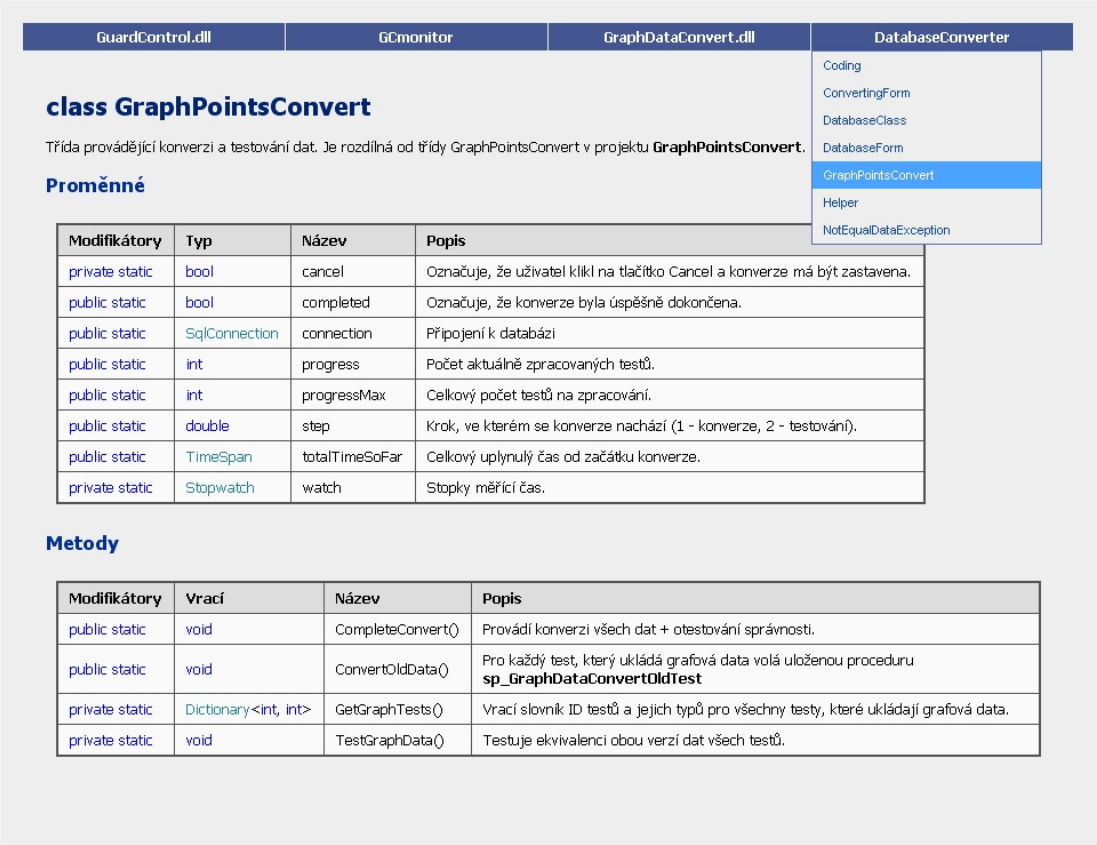
B.6 Rozdíl mezi webovou a desktopovou verzí

Obě verze skýtají stejné možnosti vykreslování, je tedy možné pomocí obou dosáhnout stejného výsledku. Desktopová verze má vykreslený graf čistější, protože nedochází ke kompresi do formátu jpeg. Webová verze nepodporuje tuto funkčnost:

- Posunování v grafu pomocí šipek doprava/doleva.
- Možnost přímého uložení obrázku do souboru či clipboardu.
- Ukládání nastavení.
- Rozšířené možnosti zoomování - lze zoomovat pouze do/ze středu grafu pomocí dvou tlačítek s lupou.

C Programátorská příručka

Programátorskou příručku jsem pro tuto práci vytvořil jako HTML stránky. Je dostupná na přiloženém CD. V horní části je hlavní menu, ve kterém je na výběr ze čtyř projektů souvisejících s touto prací. Po najetí na jednotlivé sekce se rozbalí seznam souvisejících tříd. Po kliknutí na třídu se zobrazí podrobnosti týkající se této třídy. Prostředí je vidět na obrázku 29.



Obrázek 29: Ukázka prostředí programátorské příručky

D Postup při přechodu na novou verzi - ORBIT

Konverze centrální databáze proběhne jako první v pořadí při přechodu na novou verzi. Tento krok bude zcela bezpečný, protože pouze přidá do databáze nová data a funkčnost. V databázi budou tedy souběžně obě verze grafových dat. Stará data budou odstraněna až ve chvíli, kdy všichni klienti budou používat novou verzi, a tedy budou do cetrální databáze replikovat nová data. Po dokončení konverze již bude v provozu nová verze webového rozhraní, vykreslující grafy z nových dat.

D.1 Přidání nových tabulek a funkčnosti do databáze

- Spuštěním skriptu *DatabaseChanges.sql* - ve skriptu je třeba správně vyplnit aktuální cestu k souboru *GraphDataConvert.dll*.

D.2 Konverze starých dat na nové

- Stará data z tabulky *GraphData_5m* se překonvertují na nová pomocí aplikace *DatabaseConverter.exe*.
- Tato operace může být dosti časově náročná. Konverze zkušebního vzorku dat s přibližně 91 000 000 řádky trvala 2 hodiny, další 2 hodiny pak trvalo testování správnosti dat. Použitý osobní počítač obsahuje dvoujádrový 3GHz procesor, 4GB DDR2(1066MHz) a dva 7200ot. disky SATA2 s velikostí cache 16MB.

D.3 Nakopírování nových souborů

GuardControl.dll (Typicky v C:\Program Files\visionapp Application Delivery Management Suite 2008)

- Záloha stávající verze.
- Překopírování nové verze.

gcTester.exe (Typicky v C:\Program Files\ORBIT\GuardControl\core)

- Záloha stávající verze.
- Zastavení testovací služby.
- Překopírování nové verze.
- Start testovací služby.

D.4 Nasazení nové verze webového rozhraní

- Pro novou verzi je nejdříve nutné mít na serveru nainstalovaný AJAX.
- Po konverzi je možné nasadit novou verzi webového rozhraní.

D.5 Smazání starých dat a záloh

Teprve ve chvíli, kdy všichni klienti budou mít nainstalovanou novou verzi (viz nálsedující sekce), je možné odstranit stará data z centrální databáze.

- Záloha starých dat z tabulky *GraphData_5m*.
- Nasazení gcTesteru, který již nebude volat uloženou proceduru konvertující stará data z tabulky *GraphData_5m*.
- Spuštěním skriptu *RemoveOldData.sql* se odstraní stará data z databáze.

D.6 Rollback

Rollback je možné v případě potřeby provést kdykoli před odstraněním starých dat z databáze.

- Nakopírování záloh souborů *GuardControl.dll* a *gcTester.exe* zpět na původní umístění.
- Spuštěním skriptu *DatabaseChangesRollback.sql* se databáze dostane do původního stavu.

E Postup při přechodu na novou verzi - klienti

E.1 Přidání nových tabulek a funkčnosti do databáze

- Spuštěním skriptu *DatabaseChanges.sql* - ve skriptu je třeba správně vyplnit aktuální cestu k souboru *GraphDataConvert.dll*.

E.2 Konverze starých dat na nové

- Stará data z tabulky *GraphData_5m* se překonvertují na nová pomocí aplikace *DatabaseConverter.exe*.
- Tato operace může být dosti časově náročná. Konverze zkušebního vzorku dat s přibližně 91 000 000 řádky trvala 2 hodiny, další 2 hodiny pak trvalo testování správnosti dat. Použitý osobní počítač obsahuje dvoujádrový 3GHz procesor, 4GB DDR2(1066MHz) a dva 7200ot. disky SATA2 s velikostí cache 16MB.

E.3 Nakopírování nových souborů

GuardControl.dll (Typicky v C:\Program Files\visionapp Application Delivery Management Suite 2008)

- Záloha stávající verze.
- Překopírování nové verze.

gcTester.exe (Typicky v C:\Program Files\ORBIT\GuardControl\core)

- Záloha stávající verze.
- Zastavení testovací služby.
- Překopírování nové verze.
- Start testovací služby.

E.4 Smazání starých dat a záloh

- Nejprve je třeba se přesvědčit, že nová verze bez problémů funguje (zobrazit a vyzkoušet nějaký graf).
- Stará data není třeba zálohovat, k záloze dojde z centrální databáze.
- Spuštěním skriptu *RemoveOldData.sql* se odstraní stará data z databáze.
- Možnost odstranit zálohy starých verzí *GuardControl.dll* a *gcTester.exe*.

E.5 Rollback

Rollback je možné v případě potřeby provést kdykoli před odstraněním starých dat z databáze.

- Nakopírování záloh souborů *GuardControl.dll* a *gcTester.exe* zpět na původní umístění.
- Spuštěním skriptu *DatabaseChangesRollback.sql* se databáze dostane do původního stavu.